(17+1)

PROCEEDINGS OF THE SYMPOSIUM ON INFORMATION STORAGE AND RETRIEVAL

(NASA-CR-128281) PROCEEDINGS OF THE SYMPOSIUM ON INFORMATION STORAGE AND RETRIEVAL J. Minker, et al (Maryland Univ.) 1971 282 p

N72-75387 thru N72-75404 Unclas 00/99 39468

SPONSORED BY:

The Association for Computing Machinery Special Interest Group On Information Retrieval (SIGIR)

The National Aeronautics and Space Administration

The University of Maryland

EDITORS:

Jack Minker, University of Maryland

Sam Rosenfeld, National Aeronautics and Space Administration

April 1–2, 1971

Reproduced by NATIONAL TECHNICAL INFORMATION SERVICE US Department of Commerce Springfield, VA. 22151

282

Copyright© 1971 by the Association for Computing Machinery

All rights reserved. No part of this book may be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording or by any information retrieval system, without permission in writing from the:

ASSOCIATION FOR COMPUTING MACHINERY
1133 Avenue of the Americas
New York, New York 10036

Printed in United States of America

PREFACE

In organizing this Symposium, we have had the invaluable assistance of several people. First, it is doubtful that we could have organized the conference and developed the proceedings without the advice and help of Mr. Edwin A. Crispin, Conference Coordinator, Conferences and Institutes Division, University College, University of Maryland, and his secretarial assistant, Mrs. Caroline V. Grau. Second, we thank the session chairmen, Dr. Harold Borko, Dr. Harold Edmundson, Mr. John Gosden, Dr. Michael Lesk, and Dr. Michael Senko, for serving as reviewers and for their suggestions in improving the technical quality of the Symposium.

We gratefully acknowledge the support given to us by our respective organizations, the University of Maryland and the National Aeronautics and Space Administration. The Special Interest Group on Information Storage and Retrieval (SIGIR) and the Association for Computing Machinery have also supported this activity and publicized our efforts.

Jack Minker
Samuel Rosenfeld

CONFERENCE COMMITTEE ROSTER

GENERAL CHAIRMAN/PROGRAM CHAIRMAN

Dr. Jack Minker
Computer Science Center
University of Maryland

Samuel Rosenfeld
NASA, OART
Washington, D.C.

TECHNICAL PROGRAM

Dr. Harold Borko
School of Library Science
University of California at Los Angeles

Dr. Harold P. Edmundson
Computer Science Center
University of Maryland

John Gosden
Equitable Life Insurance
New York, New York

Dr. Michael Lesk
Bell Telephone Laboratories
Murray Hill, New Jersey

Dr. Michael E. Senko
IBM Research, IBM Corporation
San Jose, California

ADMINISTRATIVE COORDINATOR

Edwin A. Crispin
Conferences and Institutes Division
University of Maryland

SECRETARY

Mrs. Caroline V. Grau
Conferences and Institutes Division
University of Maryland

Preceding page blank

TABLE OF CONTENTS

Page

Preface iii

Conference Committee Roster v

INTRODUCTION AND PERSPECTIVES FOR THE 1971 ACM IS&R SYMPOSIUM 1

SESSION 1. 9:00 - 11:00 Thursday, April 1, 1971

NATURAL LANGUAGE PROCESSING AND QUERY SYSTEMS

Chairman: Jack Minker, University of Maryland

1. The Function of Semantics in Automated Language Processing
   Milos Pacak and Arnold W. Pratt, Division of Computer Research and Technology, National Institutes of Health, Bethesda, Maryland 5

2. How Features Resolve Syntactic Ambiguity
   Bozena H. Dostert and Frederick B. Thompson
   California Institute of Technology, Pasadena, California..... 19

3. The Converse Natural Language Data Management System: Current Status and Plans
   Charles Kellogg, John Burger, Timothy Diller and K. Fogt
   System Development Corp, Santa Monica, California 33

SESSION 2. 11:15 - 12:45 Thursday, April 1, 1971

NATURAL LANGUAGE IN DOCUMENT RETRIEVAL SYSTEMS

Chairman: Harold Borko, University of California at Los Angeles

1. Cue: A Preprocessor System for Restricted, Natural English
   David B. Loveman, John A. Moyne and Robert G. Tobey
   IBM, Cambridge, Massachusetts..... 47

2. Full Text Document Retrieval: Hebrew Legal Texts
   Y. Choueka, M. Cohen, J. Dueck, A. Fraenkel, and M. Slae
   The Weizmann Institute of Science
   Rehovot, Israel 61

SESSION 3. 2:15 - 3:45 Thursday, April 1, 1971

THEORETICAL CONCEPTS

Chairman: Harold P. Edmundson, University of Maryland

1. Quantification in Query Systems
   John L. Kuhns, Operating Systems Inc, Encino, California 81

2. The Relational Data File and the Decision Problem for Classes of Proper Formulas
   Robert A. DiPaola, The Rand Corp., Santa Monica, California 95

SESSION 4. 4:00 - 6:15 Thursday, April 1, 1971

DATA MANAGEMENT SYSTEMS

Chairman: John Gosden, Equitable Life Insurance, New York, N.Y.

1. Managing Semantic Data in an Associative Net
   Stanley Y.W. Su, Electrical Engineering Dept., University of Florida
   Gainesville, Florida 105

Preceding page blank

Page

2. A Heathkit Method for Building Data Management Programs
Earl Hunt and Gary Kildall, U.S. Navy Postgraduate School
Monterey, California 117

3. File Structure Determination
Anthony J. Winkler and Alfred G. Dale, United States Air Force Academy,
Colorado, and the University of Texas at Austin, Texas 133

SESSION 5. 9:00 - 12:00 Friday, April 2, 1971

FILE ORGANIZATION AND EVALUATION

Chairman: Michael E. Senko, IBM Research, IBM Corp.,San Jose, California

1. An Approach to Research in File Organization
Daniel Teichroew, University of Michigan, Ann Arbor, Michigan 147

2. Quantitative Evaluation of Design Trade-Offs in File Systems
C.P. Wang and V.Y. Lum, IBM Research Laboratory, San Jose,
California 155

3. Elements of the Randomized Combinatorial File Structure
Richard A. Gustafson, USAF VELA Seismological Center
Alexandria, Virginia 163

4. A Balanced Tree Storage and Retrieval Algorithm
Gary D. Knott, Division of Computer Research and Technology
National Institutes of Health, Bethesda, Maryland 175

SESSION 6. 1:30 - 3:30 Friday, April 2, 1971

OPTIMIZING METHODS

Chairman: Michael Lesk, Bell Telephone Laboratories, Murray Hill, N.J.

1. Efficient Utilization of Limited Access Archival Storage in a Time Shared Environment
Alan M. Katcher, IBM Thomas J. Watson Research Center
Yorktown Heights, N.Y..... 197

2. Data Compression Techniques for Economic Processing of Large Commercial Files
James E. Mulford and Richard K. Ridall, Ridall Associates, Inc.
Paoli, Pennsylvania 207

3. Optimal Classification and Its Consequences
David M. Jackson, Department of Computer Science
Cornell University, Ithaca, New York 217

SESSION 7. 3:45 - 5:45 Friday, April 2, 1971

THE FUTURE OF INFORMATION RETRIEVAL: A PANEL DISCUSSION (No Papers)

Chairman: Samuel Rosenfeld, NASA, OART, Washington, D.C.

Panel Discussants: Harold Borko, Harold P. Edmundson, John Gosden, Michael Lesk, Jack Minker and Michael E. Senko

INTRODUCTION TO THE KEY WORD IN CONTEXT INDEX (KWIC) 225

KWIC INDEX 227

INTRODUCTION AND PERSPECTIVES FOR THE 1971 ACM INFORMATION STORAGE AND RETRIEVAL SYMPOSIUM

Jack Minker and Sam Rosenfeld
University of Maryland, College Park, Maryland, and
National Aeronautics and Space Administration, Washington, D. C.

ABSTRACT

An introduction and some prospectives are provided for the 1971 ACM Information Storage and Retrieval Symposium held at the University of Maryland on April 1 and 2, 1971. The symposium, sponsored by the University of Maryland, the National Aeronautics and Space Administration and the Special Interest Group on Information Retrieval (SIGIR) of the ACM, focuses on advances in techniques in the computer oriented technology of information retrieval. Early developments and the status of recent efforts in document retrieval, question-answering and data management systems are reviewed briefly.

KEY WORDS AND PHRASES

document retrieval, question-answering, data management, automatic indexing, natural-like languages, syntactic analysis, semantic analysis

I. Developments in Information Retrieval

The field of information storage and retrieval may be partitioned into three areas: document retrieval systems, question-answering systems, and generalized data management systems. Although the techniques required in each of these three areas are quite similar, work in each area has been performed in isolation of the others. It is towards bringing together the common underlying methods in these areas that this symposium is directed.

To better understand our objectives with respect to the symposium, it will be well to review the developments in each of the three areas.

The modern era of information storage and retrieval is generally considered to have started with the landmark article written by Vannevar Bush (3) that appeared in 1945 in the Atlantic Monthly. The concept of a library with individuals having visual displays connected remotely to a large store by terminals was foreseen by Bush. Indeed, the concept of statistical association techniques was described implicitly in the article.

Ten to fifteen years elapsed before some of the techniques described by Bush came into being. The use of computers for information retrieval started in earnest in the late 1950's. It was during this time that the Key Word in Context Index (KWIC) was developed by H. Peter Luhn (10), and independently, and perhaps earlier, by a group at Rocketdyne Corporation (4). Work in statistical analysis of text was prominent in that time period, and culminated in 1964 with an important conference sponsored by the National Bureau of Standards (18) and devoted entirely to statistical techniques. The 1964 conference represents, perhaps, the high point of work in statistical analysis for document retrieval systems. The State-of-the-Art Report written by Mary Elizabeth Stevens (17) in 1965 surveys the major techniques in automatic indexing. The Stevens' report which was updated by adding a new section and reissued several years later, shows no great progress in the intervening years.

It was also during the late 1950's that work in question-answering systems started. A system, called BASEBALL, developed by Green et al. (8) was implemented. Syntactic analysis of English-like query statements was used in BASEBALL. The development in BASEBALL led to the recognition of the importance of natural-like language inputs to a computer for query systems.

The first conference sponsored by the ACM in information storage and retrieval was held in 1961 in Princeton, New Jersey (7), with Jack Minker and Mandalay Grems as co-chairmen. Several important papers appeared at that conference. Among them were the string manipulation language, COMIT, developed by Yngve (19), and the Cheatham and Warshall paper (5) on techniques to translate retrieval requests couched in a semi-formal English-like language.

The technology of generalized data management systems also started in the late 1950's at a number of places (11). Some of the early work was done in the government at the David Taylor Model Basin, and in industry at the General Electric Company and RCA. The major techniques in this technology were developed during that time, although they were not made available to other researchers who subsequently had to reinvent the technology themselves.

The 1960's, in contrast to the flurry of developments in the late 1950's was somewhat disappointing as far as the technology of information storage and retrieval was concerned. One would have hoped for the achievement of operational document retrieval systems used by a wide variety of individuals. However, only a handful of large government organizations, or government sponsored

organizations, have implemented document retrieval systems. Notable among these is the National Library of Medicine's MEDLARS effort (2) that became operational in the second half of the 1960's.

The work by Salton and his colleagues on the SMART system (14) attempted to provide information as to the effectiveness of alternative methods that had been proposed for automatically indexing text. The results are summarized in Salton's book (14). However, many questions still remain unresolved since the text samples that were used were rather small relative to practical problems.

A conference on the Intrex Project (13), a large automated library research effort at the Massachusetts Institute of Technology, was held in the mid 1960's. The Intrex project has adopted many of the objectives originally set forth by Vannevar Bush. However, few technical papers appear in the literature on Project Intrex. A progress report on Project Intrex was presented in a series of papers at the 1969 Spring Joint Computer Conference (1). The effort still remains in the research stage, and no dramatic advances have been evidenced by the work.

The technology of question-answering systems, surveyed so optimistically by Simmons in 1965 (15) and again in 1970 (16), still remains in the research stage. No operational question-answering systems have been developed, nor are any expected to be developed within the near future. In another survey of the technology, Minker and Sable (12), note that the reasons for this are not because of fundamental limitations in computer hardware or software technology, but becuase of fundamental gaps in our intellectual knowledge of syntactic and semantic analysis of natural language, and in effective search procedures for performing inferences by mechanical means.

One possibly bright area in the 1960's was the development of a large number of generalized data management systems (11). The CODASYL Committee surveyed several such systems in 1969 (6). And yet, the techniques employed by the most advanced systems were no more advanced than those described by the early systems. It took almost ten years to recognize the importance of the technology.

Perhaps one of the most glaring disappointments in the 1960's is the lack of effective systems for library automation, and the paucity of such systems. One of the world's largest libraries, the Library of Congress, although having sponsored a major study on automation of its functions (9), lies mainly untouched by computers. The call for networks of libraries seems to be currently rampant although the development of economical systems for small libraries has yet to be attained. Fundamental processing techniques for work in information storage and retrieval either do not yet exist, or are not generally available. Indeed, advanced textbooks have not been written for either the area of question-answering systems or generalized data management systems. Salton's textbook (14) is a pleasant contrast, in that it makes available many of the major techniques currently applicable in document retrieval.

II. The 1971 Information Storage and Retrieval Symposium

In this symposium, we focus on advances in techniques in the computer oriented technology of information retrieval. It is our contention that unless the fundamental processing techniques are developed, and made available to workers in the field, information retrieval will continue to flounder, as we believe it did in the 1960's. Hence, although many papers were received for this symposium concerning overviews of interesting systems, only those papers in which detailed algorithms, procedures, heuristics, evaluative methods or theoretical concepts were described have been accepted.

Two issues are perhaps common threads of this symposium: specifying natural-like languages for the user to be able to formulate questions, and designing effective and efficient information systems. Of course, these issues provide, in some sense, different perspectives of the same problem. The specification of a user language provides an outside view of an information system, whereas the design provides the inside view of such a system. The technologies meet and overlap. Neither the language designer nor the question-answering designer can ignore data management and file organizations. Nor can the file organizer ignore the nature of the query language. In both cases, however, general techniques are required to handle classes of problems rather than be restricted to only one application. The tradeoff problems in file organization techniques are explored in the session "An Approach to Research in File Organization," chaired by Dr. Michael Senko. Some optimization methods that may be used to save space on peripherical devices are developed in the session chaired by Dr. Michael Lesk, "Optimizing Methods." Techniques developed for answering questions both for document retrieval and question-answering systems are covered in the sessions chaired respectively by Dr. Harold Borko, "Natural Language in Document Retrieval Systems," and Dr. Jack Minker, "Natural Language Processing and Query Systems." Theoretical issues that have arisen in question-answering systems, are explored in the session, "Theoretical Concepts," chaired by Dr. H. P. Edmundson. Finally, developments in evaluating and developing generalized data management systems are covered in the session, "Data Management Systems," chaired by Mr. John Gosden.

We hope that future conferences sponsored by

Special Interest Group on Information Retrieval (SIGIR) of the ACM will continue to stress theoretical results; optimization methods; and algorithmic, heuristic, and computational methods in information storage and retrieval as we have tried to do in this symposium. It is our hope that through such efforts, greater progress may be achieved during the 1970's than has been achieved in the 1960's. Perhaps then we will be able to come closer to achieving Vannevar Bush's dream.

REFERENCES

(1) AFIPS CONFERENCE PROCEEDINGS, vol. 34; 1969 Spring Joint Computer Conference, Boston, Massachusetts, May 1969, p. 457-490.

(2) AUSTIN, C. J. The medlars system. Datamation, 10:12 (December 1964) p. 28-31.

(3) BUSH, VANNEVAR. As we may think. The Atlantic Monthly 176, (July 1945) p. 101-108.

(4) CARLSEN, R. D.; GERNER, N. H.; MARSHALL, H. S. Information control. Industrial Engineering Department 564, Ref. 11, Rocketdyne Div., North American Aviation, Canoga Park, California. (August 1958).

(5) CHEATHAM, T.; WARSHALL, J. Translation of retrieval requests couched in a "semiformal" English-like language. Communications of the ACM, 5:1 (January 1962) p. 36-38.

(6) CODASYL SYSTEMS COMMITTEE TECHNICAL REPORT. A survey of generalized data base management systems. Association for Computing Machinery, May, 1969.

(7) COMMUNICATIONS OF THE ACM, 5:1 (January 1962).

(8) GREEN, B. F., JR.; WOLF, A. K.; CHOMSKY, C.; LAUCHERY, K. BASEBALL: An automatic question-answerer. In: Proceedings of the Western Joint Computer Conference, 1961, vol. 19, p. 219-224.

(9) KING, GILBERT; et al. Automation and the Library of Congress. Library of Congress, Washington, D. C., December, 1963.

(10) LUHN, HANS PETER. Key-Word-In-Context idea for technical literature (KWIC Index). Presented at American Chemical Society, Division of Chemical Literature at Atlantic City, N. J., 14 September 1959. Report No. RC127, International Business Machines Corp., Yorktown Heights, N. Y., 30 September 1959.

(11) MINKER, JACK. Generalized data management systems - some perspectives. University of Maryland, Computer Science Center, TR69-101, December 1969.

(12) MINKER, J.; SABLE, J. D. Relational data system study. RADC TR-70-180, Final Technical Report, Rome Air Development Center, September 1970.

(13) PLANNING CONFERENCE ON INFORMATION TRANSFER EXPERIMENTS (INTREX). Woods Hole, Massachusetts, 2 August - 3 September 1965. Report: Overhage, Carl F. J.; and Harman, R. Joyce (eds.). MIT Press, Cambridge, Mass. (1965) 276 p.

(14) SALTON, G. Automatic information organization and retrieval. McGraw-Hill, Inc., New York, 1968.

(15) SIMMONS, R. F. Answering English questions by computer: a survey. Communications of the ACM, (January 1965) p. 53-70.

(16) SIMMONS, R. F. Natural language question-answering systems. Communications of the ACM, 13:1 (January 1970) p. 15-30.

(17) STEVENS, MARY ELIZABETH. Automatic indexing: a state-of-the-art report (revised ed.). U. S. Department of Commerce, National Bureau of Standards, NBS Monograph 91, February 1970, 290 p.

(18) STEVENS, M. E.; GIULIANO, V. E.; HEILPRIN, L. B. Statistical Association methods for mechanized documentation. United States Department of Commerce, National Bureau of Standards, Miscellaneous Publication 269, Washington, D. C. 1964.

(19) YNGVE, V. H. COMIT as an IR language. Communications of the ACM, 5:1 (January 1962) p. 19-28.

THE FUNCTION OF SEMANTICS IN AUTOMATED LANGUAGE PROCESSING

Milos Pacak and Arnold W. Pratt
Division of Computer Research and Technology
National Institutes of Health
Bethesda, Maryland

ABSTRACT

This paper is a survey of some of the major semantic models that have been developed for automated semantic analysis of natural language. Current approaches to semantic analysis and logical inference are based mainly on models of human cognitive processes such as Quillian's semantic memory, Simmon's Protosynthex III and others. All existing systems and/or models, more or less experimental, were applied to a small subset of English. They are highly tentative because the definitions of semantic processes and semantically structured lexicons are not formulated rigorously. This is due mainly to the fact that it is unknown whether a unique, consistent hierarchization of the semantic features of language is possible.

However, the models described are significant contributions to an unexplored field called semantics. The progressive development of a sophisticated, semantically based system for automated processing of natural language is a realistic goal. It should not be neglected, despite the fact that it is difficult to predict when this goal will be achieved.

KEY WORDS AND PHRASES

natural language processing, grammars, semantics, computational linguistics

I. Introduction

Until 15 years ago, the principal attention of linguists was focused on two levels of linguistic analysis, namely phonology and morphology. Even syntax was treated rather summarily, and semantics, i.e., the problem of meaning and reference, was considered by linguists to be something not quite amenable to scientific treatment.

However, the same was not true of philosophers. Among the notable modern philosophers whose contributions are significant for the development of formal semantics are Russell and Wittgenstein, one of the founders of the Vienna Circle Positivist School. Positivists are known mainly for their proposal of a criterion of factual meaningfulness which is based on the separation of meaningful and meaningless assertions. According to this criterion, a statement is factually meaningful if and only if it can be verified by some sense experience. Wittgenstein and his followers attempted to elicit meaning from a careful examination of how words are actually used in everyday speech and writing.

The use of computers for natural language data processing, mainly for machine translation, brought the attention of linguists, logicians, information scientists, computer scientists and others to the problem of semantics. It was realized that even the most sophisticated procedures for morphological and syntactic analysis are not sufficient for the description of language. When we speak about computational morphology[3,4,5,6] (automated segmentation of word forms) and computational syntax[7,8,9] (automated description of sentence structure), we cannot ignore the aspects of computational semantics (the automated description of the semantic structure of a sentence), without which a semantic interpretation of an utterance would be incomplete. What is the definition of computational semantics? Bar-Hillel proposed two definitions,[10] namely:

"Using computers as an aid for producing semantic theories of natural language" and "using computers for analysis of the semantic structure of sentences." Bar-Hillel rejected the first definition but conceded that computers might be useful for the description of the semantic structure of sentences. There is no doubt that computational semantics can expose hidden ambiguities which often are not readily apparent to a human being, since human beings may interpret an utterance in one particular way and often are unaware of the intended meaning or other interpretations, even within the whole context.

Computational approaches to semantics are based mainly on translating natural language into predicate-calculus formulas, relational structures, or statements in a formal procedural language. None of these approaches have produced an adequate semantic theory, but all contribute towards the formulation of a possible theoretical basis.

A semantic theory of natural language is part of a linguistic description of that language. It is a theory of a speaker's ability to interpret the sentences of his language. A linguistic description generally consists of three (3) parts: phonology, grammar (morphology and syntax), and

ACKNOWLEDGMENT

The authors express their acknowledgment to G. Dunham, M. Epstein and H. Howerton for their valuable comments of the preliminary draft of this paper. Their suggestions are reflected in the paper.

Preceding page blank

semantics (lexicon). Although substantial progress has been made in the field of speech analysis and synthesis, and in morphology and syntax, the real problems arise in the area of semantics, where the lexicon must be characterized and the nature of meaning and reference must be explained.

The basic steps which are usually associated with the processing of natural language include:

(a) A dictionary look-up of lexical components and additional associated information (morphosyntactic and semantic properties).
(b) A syntactic analysis which is a formal description of the sentence structure.
(c) A semantic analysis implies the transformation of syntactic structures into unambiguous entities in a formal language.
(d) A semantic interpretation asserts the functional associations and relations between objects as perceived by humans.
(e) An inference procedure for the recognition of syntactically distinct but semantically equivalent structures.
(f) A syntactic-semantic generative system which produces English statements from certain formal structures.

The contemporary, linguistically-oriented attempts to formalize semantics are based on Chomsky's[11] theory of transformational grammar. In transformational grammar the base component generates deep structures which receive a semantic interpretation and are mapped by transformation rules into a surface structure. The resulting surface structure is then mapped into a representation of the speech signal by the application of phonological rules.

A brief description of some of the major semantic theories developed during the last ten years will now be outlined.

II. Description of Semantic Models

One of the most discussed semantic theories which is based on transformational grammar, is that developed by Katz, Fodor, and their collaborators.[12,13] Their criteria for the development of a semantic theory can be summarized as follows:

(a) The theory must show when a sentence has no congruous semantic interpretation ("He painted the walls with silent paint.")
(b) When a sentence is ambiguous ("The bill is large.")
(c) When two sentences are synonymous.
(d) When one part of the sentence is in contradiction with another part.
(e) When the truth value of a statement can be determined from the knowledge of the language, and when it is determinable only from knowledge of the world.

The abstract form of a semantic theory consists of the following components:

(a) A theory of a function F
(b) Arguments:
  (1) Sentence, S
  (2) Grammatical description of S, GS
  (3) Semantic interpretation of S, IS
  (4) Abstract characterization, C

Consequently a semantic theory is represented as a quadruple: (S,GS,IS,C)

Components of the Katz-Fodor (K-F) Semantic Theory:

A semantic theory must contain two basic components:

(a) Lexicon (dictionary)
(b) System of rules (projection rules) which operates on the full grammatical description of a sentence and on the dictionary entries. The projection rules should produce semantic interpretations for every sentence in a language and must account for a speaker's ability to understand sentences, mark semantic ambiguity and relate sentences which are paraphrases of each other.

The function of the dictionary is to provide a characterization of every sense which is conveyed by the lexical entry in any natural language sentence. The reason why the lexicon must be accounted for as a component of a semantic theory is that some sentences which differ only morphemically are interpreted as different in meaning (The tiger/mouse bit me), while other sentences which are different on the morphological level are interpreted as identical in their meaning ("Two chairs are in the room."; "There are at least two things in the room, and each is a chair."). Thus, the combination of lexicon and projection rules provides us, in theory, with the selection of the appropriate meaning of a sentence.

The Structure of Dictionary Entries:

The information which should be provided for each lexical entry consists of two parts:

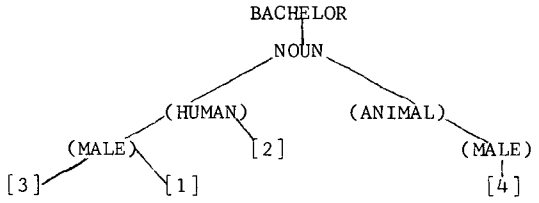
(a) Grammatical characterization (part-of-speech class membership)
(b) Semantic characterization, i.e., the senses which are exhibited by the lexical entry according to its part-of-speech class membership. For example, four dictionary definitions for the word 'bachelor' are given:

BACHELOR - NOUN
(1) A young knight serving under the standard of another knight.
(2) One who possesses the first (or) lowest academic degree.
(3) A man who has never married.
(4) A young fur seal without a mate during the breeding time.

However, K-F believe that the dictionary representation in the above form is not adequate for a semantic theory. They propose the following representation for 'bachelor': (see the chart on the following page)

The unenclosed items are called grammatical markers, items in parenthesis semantic markers (relational concepts), and items in brackets are distinguishers. The function of semantic markers and distinguishers is to atomize the lexical meaning of a lexical entry in such a way that the

semantic relations between dictionary entries are clearly stated. The semantic relations among various senses of different lexical stems are represented by formal relations between semantic markers and distinguishers. The distinguisher must appear as a terminal element, i.e., there must be no branching under a distinguisher. If branching under a distinguisher were permissible, the theory will postulate at least one semantic relation which its dictionary fails to represent by semantic markers. The function of distinguishers is not clearly defined. It is not stated explicitly whether and under which conditions the sequence of semantic markers, which is unordered, should be followed by distinguishers. It seems that a decision will presuppose a dictionary definition of features which are merely sorted into markers and distinguishers.



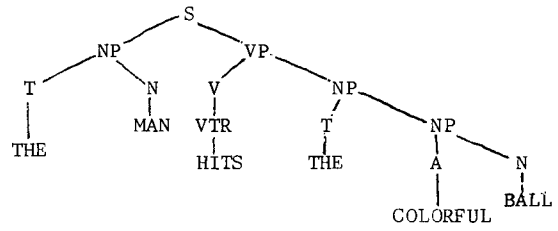
In addition to the notions of "semantic markers" and "distinguishers," K-F suggested the usage of so-called selection restrictions, i.e., the conditions under which the given sense may combine with another sense meaningfully. They have given the following reading for one sense of the dictionary entry "honest"→Adjective→(evaluative)→(moral)→[innocent of illicit sexual intercourse] [(human) and (female)] where parentheses enclose semantic markers, brackets the distinguishers, and parentheses plus brackets enclose the selection restrictions. This says that the adjective "honest" receives the interpretation (evaluative)→(moral)→[innocent of illicit sexual intercourse] if the head noun it modifies carries the semantic markers (human) and (female).

Katz and Fodor did not discuss in their paper the morphological form of the lexical entry - segmented versus unsegmented - to be entered in the dictionary. They mentioned only that it might be useful to enter certain prefixes which exhibit certain semantic characteristics as single entries and then combine them with the unprefixed lexical form.

Projection Rules:

As mentioned, the structural description of a sentence provides the input to a semantic theory. The structural description of a sentence consists of a set of grammatical markers--as opposed to semantic markers--which specify the syntactic relations among the grammatically defined elements of a sentence. Semantic projection rules are applied to compound constituents in the deep structure which are in one-to-one correspondence with the phrase structure rules. For example, the readings of an adjectival phrase NP→A + N are derived by a rule which operates on the Boolean union of the semantic markers of the two constituents A and N, the selectional restrictions of the N, and the concatenation of the two distinguishers if the semantic markers of N agree with the selectional restrictions of the A. The set of readings for a sentence is derived from the sets of readings of the constituent lexical units by applying the projection rules recursively, item by item.

For example, the sentence "The man hits the colorful ball" is represented as follows:



The semantic interpretation of the sentence above consists of a set of combinations of semantic markers which are conjoined according to syntactic rules.

The application of projection rules yields four semantic interpretations one of which reads as follows:

The [Some contextually definite]
man (Physical object)→(Human)→(Adult)→(Male)
hit (Action)→(Instancy)→(Intensity)→
[Collides with an impact]
the [Some contextually definite]
colorful ball (Physical object)→(Color)→
[[Abounding in contrast or
variety of bright colors]→
[Having globular shape]]

The other semantic interpretations differ in the function of distinguishers for 'ball' (solid missile for projection of engine of war) and 'hit' (strikes with a blow or missile).

The result of the implementation of the projection rules conveys the following semantic interpretation: the sentence is not semantically anomalous and it is four ways semantically ambiguous.

The example given by Katz demonstrates how a semantic theory of English might interpret a sentence which is first generated by the grammar. The function of grammatical markers is to show the types of syntactic relations among the sentence-forming elements, whereas the function of semantic markers is to interpret each well-formed string in terms of its conceptual content. Consequently, Katz considers semantic markers to be theoretical constructs which are distinct from grammatical markers. A set of projection rules prescribes the procedure for continuing the readings of a pair of words or phrases to obtain a reading for a longer string which is a sentence. Katz's semantic theory does not appear capable of handling strings which go beyond sentence boundaries. The number of examples in Katz's paper is not large enough for a more detailed evaluation, but one of the major problems is the lack of discovery procedures for obtaining semantic markers. The only results that have been obtained since the first publication of the Katz and Fodor paper are refinements of the original theory. But no list of markers, selection restrictions, or projection rules has

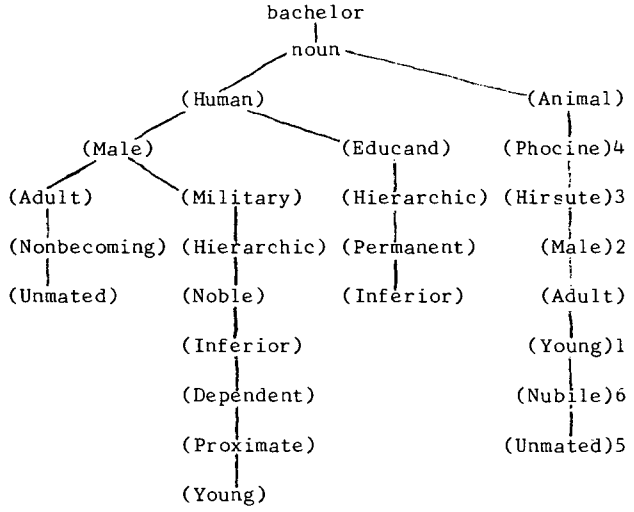
appeared since .

Despite the fact that K-F theory is not fully adequate for the description of complex semantic processes, it was a significant attempt to formalize a theory of semantics. It provided a set of goals to be achieved in the formalization of a theory, and thereby promoted research in this direction.

It should be noted that machine translation researchers were attempting to resolve semantic ambiguities in much the same way prior to Katz. Zelling Harris[14] wrote an article on the componential analysis of Hebrew which contains many features which are similar to Katz's semantic markers.

Bollinger[15] in his review of K-F theory pointed out that the distinction between semantic markers and distinguishers is not clear.

He pointed out that the distinction between "markers" and "distinguishers" was dubious and that there were no formal criteria in the theory for such distinction. For example, the dictionary entry for "BACHELOR" may have the following structure:



The numbers on the right indicate the order in which the markers should appear in the dictionary definition. The markers in K-F dictionary are unordered, although a hierarchical structure is present. Deeper subcategorization of semantic markers will be necessary; for example, the semantic unit $BALL_1 \rightarrow$(gala affair) and $BALL_2 \rightarrow$(Spherical object) will have to be provided with additional suitable markers such as 'Event' and 'Object' in order to distinguish the sentences: "He observed the ball."; "He attended the ball."; and, "He bounced the ball."

Weinreich (see below) raised similar objections to the K-F theory. In a revision of the theory, Katz abandoned the notion of "distinguisher" altogether.

In our opinion, the K-F semantic markers correspond to sets of defined semantic properties which are conveyed by the given lexical unit. The semantic properties of the given lexical entry are inherent parts of the unit. They should be ordered in such a way that the computer program will test for the occurrence of certain types of semantic properties and for semantic correlations with properties of adherent semantic units. If more specific markers are introduced, the number of markers will be increased substantially and so will the number of projection rules. The correlation rules must not only map the markers of all senses but show how markers are added and subtracted to alter the senses of words.

At the present time, there is no theory which would specify the order of semantic features in a path. The constituent paths are unordered sets and there is no way of predicting that a certain path should precede or follow some other path. For example, the path for 'detective' plus 'woman' would lead to constructions 'woman detective' and 'detective woman' since the order of elements in the path is immaterial.

The weakest point of K-F theory is the assumption of minimal lexical unit which is not defined. A K-F dictionary would have to represent an unlimited differentiation of meaning to account for idioms and synonyms. Another weakness of K-F semantic theory is the conception that the theory is composed of a dictionary and projection rules. It seems that semantic rules that describe the meanings of terms and phrases and the connections which hold between them cannot be handled by dictionaries alone. An example is the transitive expression "is colder than" which means that if A is colder than B, and B is colder than C and, consequently, A is colder than C, this cannot be inferred from the dictionary entry "cold". However, it does not mean that the importance of a semantically structured dictionary for the semantic analysis should be underestimated. An example of a dictionary which is suitable for the semantic analysis of pathology reports is The Systematized Nomenclature of Pathology. A description of how this dictionary (with certain modifications) can be used for semantic analysis of medical English will be given later. But we must stress that dictionary-type rules and paraphrastic rules are only part of the totality of semantic rules which can be of many additional types, and the number of which is undefined.

Weinreich[16] improved some of the deficiencies in the K-F theory by introducing special rules for the elimination of tautological statements, contradictions and semantic misinterpretation of metaphoric expressions. Weinreich also provided structured entries for complex idioms and developed a procedure for the computation of a quantitative measure of the deviance of a sentence from normality.

One of the weakest points in the Weinreich theory is the assumption that the binary features constitute an adequate apparatus for the representation of meaning; also semantic processes and dictionary definitions should be formulated more rigorously. Both K-F and Weinreich theories treat sentences in isolation and disregard the function of discourse and situational context in the determination of meaning.

Another approach to the solution of semantic problems is that which Garvin suggests.[18] A semantic code must be able to refer every dictionary entry to its appropriate semantic class. In other words, the semantic code should be based on

systematic classification of the elements with which it deals. He indicates that there exist synonym lists and thesauri but these have been compiled intuitively and do not really represent the natural language with which they deal. Garvin suggests we pattern a semantic theory along the lines developed for syntactic theories. His basic assumption would be a system of meanings for languages just as there exists a system of syntactic forms. If we assume this, then we can apply linguistic methods to the problem of meaning. If we want actually to observe linguistic meaning in action, we can do it either through translation results or paraphrasing. According to Garvin, these are two possible directions of research and we can learn much from the application of the technique of paraphrasing and the investigation of the role of context in definition of meaning. The paraphrasing technique he used was based on the idea of replacement of predicates or the replacement of the verbal elements in a sentence. If one applies replacement of predicates successively one finds that a series of semantic features for each verbal element has occurred. On the basis of differences and similarities in their semantic features, one can arrange the verbal elements in a thesaurus which is inductively derived from a processing of text. As soon as the verbal elements have been arranged in the thesaurus, one can take the nominals and on the basis of co-occurrence with semantic classes of the verbal elements, each nominal will be assigned its semantic features depending on whether it is subject or object of the members of the various classes of the thesaurus.

Garvin[19] chose predication typing as the basis for a pilot study in semantic analysis. His basic assumption is that it is possible to establish patterns of similarities and dissimilarities in meaning for different types of linguistic units. Once the patterns are established a method for semantic classification can be developed. The technique he used for predication typing was to have human subjects paraphrase the sentences of a given text with a set of specific instructions. Paraphrasing has been defined as an operation by which a statement or a part of it is replaced by another statement while the original meaning is retained.

The basic conditions in Garvin's discovery procedure are as follows:

(a) The original statement in the given text
(b) The choice of linguistic units which are relevant for the selection of a particular predication type ('cue-form')
(c) The replacement of the 'cue-form' by its synonymous form
(d) The paraphrasing of the original statement
(e) The result is a semantic feature which is derived from the predication type

For example:

Original statement: Our research was started according to a preliminary plan
Cue-form: start
Replacement form: stem from
Resultant statement: Our research stemmed from a preliminary plan.
Semantic feature inferred from predication type: origin

A given cue-form is assigned as many semantic features as the number of predication types that have been suggested by it in the course of processing a good sized sample of text. The cue-forms are arranged in semantic classes on the basis of their semantic spectra which contain all the semantic features of a given cue-form, modified by a weighting factor. In other words, all possible meanings of the cue-form, based on the predication types are available in the semantic spectrum.

Only 33 predication types were used in Garvin's study. They were of various types such as 'causation' (A is due to B); 'origin' (A stems from B) etc.; they allowed for manipulation of the grammatical form of the sentence so long as, in the final outcome, the criterion for the comparison property was maintained and rough similarity of meaning was maintained.

The study was done in the hope that a method for the development of semantic classification of pertinent linguistic units could be found. In the course of it, the cue-form families were determined to be those pertinent linguistic units which display similarity groupings.

Some of the obvious inadequacies in Garvin's procedure are the lack of criteria for the determination of a 'cue-form' and, consequently, of 'semantic spectrum' (collection of semantic features of the given 'cue-form') and the lack of distinctiveness between predication types.

Fillmore[20,21] considers each sentence in a language to be a unit composed of a verb and a collection of nouns in various "cases" in the deep structure sense. The term "case" is used to identify the underlying syntactic-semantic relationship among the constitutents of a sentence. Case distinctions such as agentive, objective, resultative, instrumental, locative, extension and others, are sometimes explicit and sometimes they are implicit in the surface structure depending on the noun, or on properties of certain governing words.

Fillmore's formalism corresponds essentially to the predicate-argument relationship of logic. Each argument is labeled according to its role in a predication by a case relation. Every predication includes a modality constituent containing information on tense, aspect and validity. The transformation operations include subject and object selection, selection of prepositions, or overt case forms for expression of case relations, nominalization, sequential ordering and other features which are a part of transformational grammar. Despite the fact that the list of case relations and the composition of the modality constituents is incomplete, the case grammar is powerful enough to describe concisely grammatical regularities in general English.

In a recent study, Stockwell, Schater and Parker[50] have combined Filmore's "deep case" approach with Chomsky's "lexical" hypothesis[51] into an integrated system for English. In this system, the demarcation line between syntax and semantics becomes highly tenuous.

Silvio Ceccato[22] and his group at the Center for Cybernetics and Linguistic Research at the University of Milan represent the so-called "operational school", a basic philosophical approach which can be described as follows:

The process of thinking is definable in terms of a set of mental operations. The mental operations can be reduced to four fundamental types of activity: differentiation, figuration, categorization and correlation.

Differentiation is an operation for discriminating between two things such as darkness and light, attention and disattention, etc.

Figuration provides for figures or forms. It can be described as a differentiation which occurs in places which are different in relation to a special system of reference, such as our body.

Categorization covers the mental, or logical categories including, for example, state, process, cause, effect, plural, singular, etc.

Correlation is the process of combining the results of the previously mentioned mental operations which correspond to human thinking. The resulting correlation form consists of three components: correlator, first correlatum and second correlatum. The graphic representation of data structure representing the correlation model is as follows:

CORRELATOR	
1st CORRELATUM	2nd CORRELATUM

In the first interval some mental object (idea, concept, mental image) enters the center of awareness. To communicate this object, it must be represented by a word or other symbol. In the second and third intervals a second and third mental object appear, which acts as a correlator of the other two. The roles and order of appearance of the first and second correlatum and correlator depend on meaning, and on syntax of the particular language used. The correlator functions as a relational predicate which indicates the meaningful relationship between the semantic units.

Ceccato established 56 types of relations for a dictionary of about 500 English main entries. Some relations, including those "species - genus", "part-whole" have the property of transitivity within the limits of one and the same relation. For instance, it is enough to classify "finger" as a part of "hand", "hand" as a part of "arm", "arm" as a part of "body" in order to obtain the relation part-whole between "finger" and "arm", that is between members of a group not contiguous with each other.

One hundred and six correlators (mainly prepositions and conjunctions) and their properties were described; for example, the correlational property of the preposition 'TO' exhibits three functions in spatial relationships (something reaching the object, motion in the direction of, orientation) and one function in temporal relationship (arriving point).

The basic method of attack was to analyze sentences of the source language in terms of a network of correlations. A highly organized dictionary containing the vocabulary of correlata (mainly nouns and adjectives) including subtle differentiation in usage was prepared. The analysis by computer included differentiating shades of meaning of each word by application of correlational rules to the source language and synthesis of the target language translation from the network. The detailed description of the program can be found in the final report.[22]

The correlational grammar was tested on five languages - English, German, Latin, Italian and Russian, with some promising results, but it never became fully operational. The two most limiting factors were the time and effort required for adequate construction of correlational dictionaries and the size of core memory required for storage of data. Consequently, the MT project was temporarily discontinued. However, research has continued and focused on theoretical refinements of correlational analysis.

von Glasserfelt and Pisoni[49] developed a multistore parser for hierarchical syntactic structures which is based on the correlational grammar. This grammar consists of four parts:

(a) The collection of 350 correlators, i.e., linking elements that indicate the particular semantico-syntactic relation between two items.
(b) Vocabulary which is the collection of lexical entries classified in terms of their capacity to function as components of correlations.
(c) Three hundred correlational rules by which the correlational structures are recognized and assigned correlational index number.
(d) Collection of ad hoc rules which are required by the specific application of the system, e.g., recognition of idiomatic phrases, recognition of anomalous sentences, etc.

The system which is designed to serve as the basis for further development of correlational grammar and semantic disambiguation of English sentences is experimental as are the majority of other similar systems.

Schwarcz's[23,24] approach is based also on the operationist philosophy as formulated by Benjamin;[25] it asserts that "descriptive intentions are functional operations on a data space which yield elements of knowledge as their result".

Schwarcz considered two basic processes in his semantic model, namely, decoding and encoding. Decoding consists of three parts:

(a) Syntactic recognition which includes syntactic identification of phrases and grammatical agreement among two or more constituents.
(b) Semantic translation implies interpretation functions associated with grammar rules.
(c) Application to the model representing the discourse context. At this stage, the elliptical and anaphoric constructions are to be found and semantic ambiguities resolved.

Encoding implies the transformation of a message into a well-formed surface syntactic structure through lexical substitutions and formation

of syntactic constructions.

Schwarcz suggested that encoding and decoding processes should be formulated as non-deterministic procedures employing heuristic search and evaluation techniques. Such a formulation would make it possible to introduce effectively modifications through the ability to use natural-language statements in order to make changes in these procedures and rules.

Schwarcz's approach has not been implemented. However, he claims that a semantic meta-theory that takes into account explicitly the communicational significance of language can provide a sound basis for a theory of grammar and a theory of cognition.

One of the major problems in the implementation of Schwarcz's semantic model will be the construction of an operational, semantically structured lexicon. Standard dictionaries do not always contain the kind of syntactic and semantic information which will have to be encoded in the lexicon. But once such an operational lexicon is constructed it would be usable for many kinds of applications.

Schwarcz's theoretical approach is close to Lamb's[48] stratification theory. The theory is based on the assumption that language is a finite system which is used for encoding (speech and writing) and decoding (understanding) of infinite sets of texts in the language. The relation between expression and meaning is a network consisting of four layers (strata), namely:

(a) The phonemic stratum which deals with all possible combinations of sounds in a language.
(b) The morphemic stratum deals with the structure of words and phrases.
(c) The lexemic stratum specifies the structure of clauses and sentences.
(d) The sememic stratum is concerned with the meaning of linguistic units and units larger than the sentence.

Lamb distinguishes two basic types of relations in his theory, namely: unordered "either-or" relationship and ordered "and" relationship. In an unordered type of relation the sequence of elements forming a conceptual unit is unrestricted while in an ordered relationship the order of semantic constituents is restricted and cannot be changed, e.g., in "sweet potato" the order of "potato" and "sweet" will convey different concepts.

Lamb feels that the stratification theory has the ability to predict noun phrases, to handle the formation of newly created idioms and trace the meaning (semantic interpretation) beyond the sentence boundaries.

Lamb believes that the stratification theory corresponds to human cognitive processes and shows a resemblance to the functional components of neurons in human brain. At present, this assumption has not been tested on the computer but he intends to use the computer to test the model and see whether it will produce the kinds of outputs which the brain produces. It will be of interest to compare the outputs of Quillian's semantic memory and Lamb's model to see what are the similarities and differences in both approaches.

An interesting approach to semantic analysis with strong psychological flavor is Quillian's model of semantic memory[26] and his Teachable Language Comprehender.[27] The semantic memory is a network of word-nodes which are connected by syntactic relations. The structure of semantic memory consists of various types of associative links (type-to-token, token-to-token, token-to-type), conjunctive and disjunctive sets of relations and predicate relations (subject, object relations). A type node is a root of a tree, with many branches which specify the meaning of the word at the root. A token node refers indirectly to a word concept by using special associative links that point to the concept type node. For each word there is one type node, while usually there is more than one token node.

The primary aim of the semantic operation is to compare and contrast the meaning of any two word-concepts in the memory store and generate an English statement to represent the relationship.

For example, by comparing "plant" and "live" we obtain the semantic interpretation:

(a) Plant is a live structure
(b) Plant is structure which gets food from air or water or earth.

In his most recent research Quillian has generalized his semantic network and included it in a program called the Teachable Language Comprehender (TLC).

The Teachable Language Comprehender (TLC) is a program which is designed to be capable of "comprehending" English text. The ultimate goal is the development of a computer program which will be able to extract and retain meaning from natural language text it has not seen before in the same way as human beings do.

Acting on previously unknown text TLC strives to produce an interpretation of it which relates the particular assertions of the text, both explicit and implicit, to a semantic memory containing information of a general nature about the world. The syntax or information structure of both the interpretation and of memory are identical and the resulting interpretation may include information from the program's general semantic memory, rendering explicit in terms of this memory, the assertions of the text.

The system makes use of syntactic "form tests" which are applied as a verification measure in the course of TLC's processing. Here, semantic processing drives syntactic processing.

TLC is designed to run under the tutelage of a human monitor who should supply general information about the world to the semantic memory and form tests as needed by TLC in the reading of text. Information obtained by interpretation of text may be accumulated in the semantic memory. Presumably, as this memory and the repertoire of form tests grow, the human monitor would intervene less and less. It is in this sense that TLC is teachable.

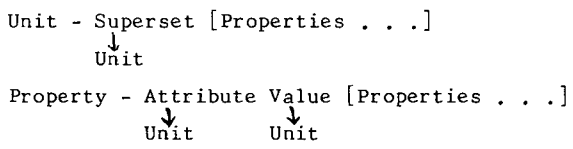
The essential features of TLC semantic memory, used also in representing an interpretation of text, are "unit" and "property". Unit is defined as a concept of an object, or an idea which is represented in English by a word, phrase or sentence. Property is any type of predication which might be a verb phrase, clause or adjectival or adverbial modifier. They consist of pointers to

units and properties resulting in an interconnected network which may be viewed as a psychological model of human performance of cognitive processes. A dictionary of English words points to one or more units representing its meaning.

A unit contains an obligatory element which is a pointer to a unit called its "superset", representing a more generic concept. Optionally, a unit may have one or more pointers to properties whose semantic function is to restrict or define the unit relative to its superset.

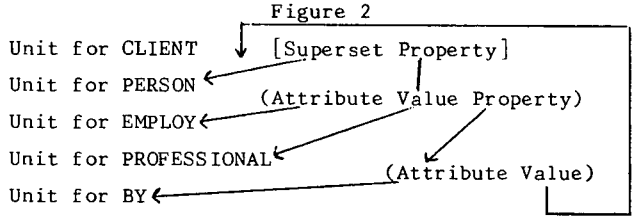
A property has two obligatory pointers to units which are called the "attribute" and the "value" of the property. Optionally a property may have one or more references to properties. [Figure 1]

Figure 1



The attribute-value pair is used for expressing any type of predication stemming from a variety of syntactic structures in English. For example, preposition and noun or verb and its object are treated as the attribute-value pair. The conceptual interpretation of the phrases 'lawyer's client' showing a client's relationship to a professional may be represented thus:

Figure 2



Quillian's algorithm for interpretation of text begins by selecting those units in the semantic memory which are immediately related to the text and creating a new unit for each word of text. TLC's algorithm then operates on these units employing semantic and syntactic processes, and encoding its interpretation as it proceeds. Presumably the encoding process will, as the algorithm proceeds, amalgamate the new units into a single structure representing the interpretation of the particular text fragment.

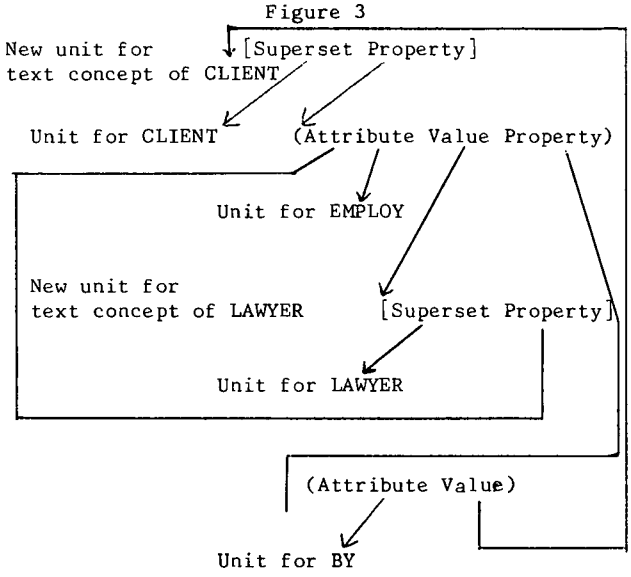
Processing is controlled by a search which yields intersections in order of presumed semantic nearness or centrality to the concepts of the units from which the network was entered.

When one of these semantic intersections is found, the two text words giving rise to the search are tested for syntactic compatibility with the tentative semantic result by means of form tests, fairly elementary syntactic criteria associated with the semantic memory. If the text words are found to be compatible in this sense, encoding of the new concept units takes place. This can involve recopying that portion of the semantic memory designated by the intersection as a property of the new unit or association of a semantic memory unit as the superset of the new unit.

The algorithm then continues to search for semantic intersections in breadth first manner upon the now enriched structures.

In interpreting the phrase "LAWYER'S CLIENT" the search finds intersection at the unit "PROFESSIONAL" which is in the superset chain of "LAWYER". This indicates that "LAWYER" may be suitable for substitution as the value of the property of "CLIENT" which has attribute "EMPLOY" and value "PROFESSIONAL". A form test validates this assumption and the result is encoded as in Figure 3.

Figure 3



The new units for the text concepts of "CLIENT" and "LAWYER" have had their supersets identified as the semantic memory concepts of "CLIENT" and "LAWYER" respectively. The property "EMPLOY PROFESSIONAL" has been copied from the semantic memory version of "CLIENT" to the new unit for the text concept of "CLIENT". The new unit for the text concept of "LAWYER" has been substituted in this property where the intersection with "PROFESSIONAL" in the superset chain of the memory unit "LAWYER" occurred.

This process leading to identification of the text concept "LAWYER" as the "PROFESSIONAL" employed by the text concept "CLIENT" demonstrates the capacity of TLC to make implicit inferences using its particular memory structure. Quillian suggests that a memory with inferential capacity is the functional equivalent of a much larger memory, and that human beings, operating in a similar way, may act as if they know a far greater amount of information than they actually need to have stored.

The basic feature of Quillian's approach is a network of encoded dictionary definitions in which pairs of words are tested to determine whether they form a predication. If it is found that they are syntactically compatible (syntactic "form test"), they are combined into a new constituent and the process continues.

There are certain problems in TLC which will have to be dealt with.

(a) Syntactic analysis will have to be improved

in order to eliminate the overinterpretation of certain types of phrases such as 'lawyer's enemy' and enemy's lawyer' which are both interpreted as 'lawyer advising an enemy in legal matters'.

(b) It is not clear whether the new property refers to the attribute only or whether it could refer to the value or to some additional property of the original unit. Our example is that the system is unable to identify a 'male child' as a 'boy'.

(c) The number of pointers to units and properties might be very large and so might be also the number of intersections. Consequently, reasonable search time is possible only at a great expense in storage capacity.

Quillian's system is psychologically oriented in much the same manner as those developed by Abelson and Colby.[28,29] All these systems are built around simulated or synthesized cognitive structures whose elements are concepts and relations, and which incorporate inference procedures as special cases of concepts. All the concept networks depend strongly on the notion of nested attribute-value lists to form computer representation for units of meaning.

However, there are serious technological problems which are connected with the construction of a rich semantic network. Operational use of natural language processing requires large dictionaries (50,000 - 100,000 entries), vast quantities of syntactic and semantic information and, possibly, storage of encyclopedias of text. One solution might be in the provision of a hundred million word cores or the development of associative computers which is not likely in the near future.

A system of interactive semantic information for disambiguation is proposed by Moyne.[52,53] He has constructed a query system for libraries and other similar applications which has the following characteristics:

The input sentence (query, command or statement) is analyzed by a transformational recognition grammar stored in a computer and a canonical deep structure is produced. This "deep structure" is the basis for the interpretation of the sentence and is used to generate calls for sub-routines and macroes which "execute" the sentence. Unlike many of the similar approaches, this system generates only one analysis (one deep structure) for each input sentence; yet a measure of generality is achieved by using the following devices in the order given below:

At the point in the derivation of the deep structure where more than one rule can apply and result in ambiguities:

(1) The data base is automatically consulted to see whether the nature of the discourse and the application at hand will rule out the application of one rule in favor of another. If this fails to resolve the ambiguity

(2) the capabilities of the underlying system are automatically examined to see whether certain idiosyncrasies of the system will preclude the application of one rule in favor of another. If this is unsuccessful, the user is asked to identify which sense of an ambiguous sentence he has in mind, before the rule is selected.

This system is claimed to be very powerful for specific applications and restricted discourse. Its obvious disadvantage is that much of the grammar has to be designed with a view for specific applications and restricted discourse. From purely a pragmatic point of view, it is still a mute question whether a restricted English for communication with a computer has advantage over artificial languages.

Kellogg[30] has developed an experimental system for compiling formal language data management procedures from a subset of natural English statements and questions. Kellogg's linguistic procedure starts with the syntactic analysis based on a context-sensitive phrase structure grammar. The lexicon contains syntactic and semantic word classes, a list of semantic features and a list of selection restrictions. Decoding is accomplished by a syntax directed procedure in which each syntactic rule is coupled with a semantic resolution rule. The resolution rules operate on semantic features and selectional restrictions which are associated with the constituents combined by the syntactic rules. Kellogg's system is very similar to Katz's semantic markers, but the markers are so far limited to the form of semantic classes and do not possess the extension structure to be required in a marker.

Some of the inadequacies of Kellogg's system are related to deviations from standard syntax, recognition of discourse structure and disambiguation by discourse context, which are weaknesses that are common to other semantic systems as well.

Another system which is semantically-oriented is so-called Protosynthex III developed by Simmons, Burger and Schwarcz.[31,32]

The base of the system is a model of human conceptual structures broken down in triples: Concept-Relation-Concept. Each concept in the triple represents the meaning of the word, and each middle term in a triple is a relational term which is not necessarily well-defined. The meaning of a sentence is a nested set of relational triples. Triples in a sentence are located by using the lexicon and grammar including transformations and tested against a stored dictionary of rules. The stored semantic rules which are called Semantic Event Forms (SEF) consist of a triple of three semantic classes. The semantic class of each word is tested in the frame "Word 'X' is a kind of --." For example, the sentence "The angry pitcher struck the umpire who called the game" is interpreted as "pitcher is a kind of game-player;" "gameplayer is a kind of person;" "angry is a kind of emotion," etc. The stored SEF rule for "angry pitcher" is represented by a triple "Person-modification-emotion" which eliminates the sense "pitcher as a container". If direct lookup for a triple is not successful a set of deductive inference rules which have the pattern-operation form ((A sister B) and (B mother C))→(A aunt C) is applied in the system.

The system was tested with success on a range of questions selected from Compton's Encyclopedia. For example, the question "Who lost the battle of

Waterloo?" was answered by a statement "Napoleon commanded the French Army which lost the Battle of Waterloo in 1815".

The system cannot answer "how" and "why" questions or adequately treat quantifications. The system was written in LISP on the Q-32 and is very slow. At present, the system is being revised with the expectation that it will operate more rapidly on larger bodies of text (5 - 10,000 words).

Woods'[33,34] method is similar to Kellogg's in that, it utilizes as the input syntactic structure which represents deep structure produced by a transformational grammar.

In his prototype system Woods introduced the notion of semantic primitives around which the system is built up. The components of semantic primitives are primitive commands by which the truth value of a proposition is tested, e. g., TEST (A); test whether A is a true proposition; primitive predicates which carry out an action, e.g., CONNECT (x1, x2, x3): flight x1 goes from place x2 to place x3, and primitive functions which compute the value of a function, e.g., Dtime (x1, x2): the departure time of flight x1 to place x2.

The semantic interpretation of a query is realized by the translation of the phrase marker into a formal representation of its meaning in terms of semantic primitives. The configurations of grammatical relations in deep-structure trees have semantic significance, if they are dominated by sentence nodes (S) and noun phrase nodes (NP) which receive semantic interpretation. The manner in which the semantic interpretation is built up of the constituents of a construction is specified by a finite set of semantic rules (approximately 80 rules). The format of semantic rules is "pattern=> action" where the "pattern" is a description of a fragment of a phrase marker and the "action" is the operation that attaches a semantic interpretation to the given fragment. There are two types of semantic rules that are used in the system, namely: S-rules which operate on S-nodes and yield propositions or commands, and NP-rules operating on NP-nodes and yielding designators. The left parts or the rules are Boolean combinations of labeled fragments of phrase markers that include specification of their terminal elements as English words or as semantic class predicates such as 'PLACE (Washington)=TRUE' implies Washington is a place. The right half of the rule is a function or predicate in the procedural language with its arguments specified as elements of the syntactic subtree of the left half. For example, CONNECT (UA-69, Washington, Miami) is interpreted as "UA-69 flies from Washington to Miami". The verb is used to determine the primitive predicate representing the meaning of an input sentence. The syntactic structure is matched against the primitive predicate, one predicate is selected, and the arguments are verified. The prepositions FROM and TO are treated as commutative and no relative order is specified. The phrase structure rules incorporate detailed specification of legitimate combinations of syntactic features and predicates as conditions of the formation of procedural language constituents. This includes tests for syntactic and semantic well-formedness to allow for selection of meanings or disambiguation of words and phrases.

A dictionary of semantic features contains entries for each verb and noun with specifications of possible semantic rules that could apply to a construction in which the given word occurs as its head. The system, as proposed, can handle universal and existential quantifiers, determiner 'the', and singular and plural nouns.

Woods' treatment of quantification is significant in comparison with the other systems. The generation of quantifiers for noun phrases consists of three parts:

(a) Determination of the type of quantifier (every, some, which).
(b) Determination of the range of quantification.
(c) Determination of additional restrictive clauses on the range of quantification.

Woods' main contribution toward the development of an adequate semantic theory is the structuring of his procedural language which is powerful enough to express in a natural way the messages conveyed in ordinary language usage. The prototype system can be expanded to include concepts which are not explicitly contained in the data base but can be deduced from it.

Woods' system also provides an interface between a semantic interpreter and the retrieval component to produce independence between semantic interpretation procedure from retrieval techniques, and the structure of the data base.

There are some weaknesses in Woods' method as well. Syntactic parsing that is independent of semantic operations implies the possibility that many syntactically acceptable but semantically non-admissible interpretations would have to be taken into consideration. It will be necessary to develop a procedure for semantic testing of syntactically admissible but semantically invalid constructions. At present, the data base is restricted to the Official Airline Guide; it remains to be seen whether the system could be expanded effectively to other contexts. There is no doubt that the expansion of the system to general English will require a substantial increase in the number of semantic primitives and, consequently, semantic rules as well.

Pratt[35] and his group have been working on the development of a semantic model for automatic processing of medical English. The domain-specific semantic model is based on the notion of cognitive structures composed of conceptual semantic units and relations among the semantic units. The lexicon base of the system is the Systematized Nomenclature of Pathology (SNOP).[36] SNOP is a special purpose lexicon created by pathologists to assist them in the organization and retrieval of information. A term or conceptual semantic unit is listed in only one of the four semantic categories and is assigned a unique numerical code within the given information class.

The four semantic categories of the SNOP are:

Topography	(T)	- the body site affected
Morphology	(M)	- the structural changes resulting from disease
Etiology	(E)	- the causative agents (microorganisms, drugs and chemicals)

Function (F) - the physiological manifestations associated with disease, including symptoms and a linited number of specific infectious diseases.

In terms of the semantic model the medical diagnosis can be defined as a semantic product of four major semantic categories T, M, E, F and a set of relational predicates R_1, R_2..., R_n. A categorial semantic unit consists of a set of elements (words) where one or more elements must belong to one of the categories. The boundaries of semantic units can be either identical or larger or smaller than the boundaries of syntactic constructs. Categorical semantic units are correlated by a set of relational predicates as mentioned above. The structure of relational predicates is determined by the co-occurrence of defined initial and terminal markers such as M + OF_1 + X + OF_2 + T where the initial marker of locative relation R-LOC is the preposition OF_1 and the terminal marker is OF_2; the intermediate element X can be either a recognizable semantic unit or it can be a non-definable unit (undefined noun i.e., the word which is not found in the dictionary) the semantic properties of which will have to be defined. The constituents of a relational predicate are different types of linguistic units (prepositions, verbs, etc.) the main function of which is the description of semantic relations among the categorial semantic units. For example, the statement "Pneumonia due to staphylococcus" is formalized as "M R-cause E" where 'R-cause'" is the causative relational predicate 'due to', "pneumonia" is the semantic unit 'M' and "staphylococcus" is the semantic unit 'E'.

The notion of semantic categories is essential for processing of medical English. Once any semantic category is specified, further definitional processes may result in a number of relations between English words and phrases and semantic categories. The semantic interpretation of a medical statement is based on the morphosyntactic and semantic information assigned to the individual words in the phrase or sentence.

It is expected that the expansion of deductive inference rules beyond the present logical framework will strengthen the precision of the semantic model and make it more general.

Montgomery[37] proposed an experimental model for analysis of natural language in terms of sets of semantically equivalent performances in particular physical and sociocultural environments. The notion of an equivalence set implies an approximate definition of relation in terms of symbolic logic. Her model provides a tool for arriving at theories of meaning through exploitation of data cases which are subsets of natural language as it is actually used in the given environment. The problems associated with the concept of equivalence-set are the definition of the basic members of the set in terms of relevant environmental features, and the definition of equivalence sets for abstract contexts. Montgomery's approach is close philosophically to the slogan of Vienna's positivist school "Do not look for meaning, but look for its actual usage"

Despite the progress which has been made in automated language processing during the past several years, there are still many syntactic and semantic problems which have to be resolved.

Some of the major difficulties are as follows:

(a) The problem of homographs,[38,39] abbreviations and acronyms
(b) The problem of synonyms, near-synonyms and antonyms
(c) The treatment of punctuation marks, mainly of commas
(d) The recognition and interpretation of anaphoric, elliptical and idiomatic constructions
(e) The problem of indirect references, including the use of pronouns
(f) The recognition of semantically equivalent but syntactically distinct constructions and their paraphrasing
(g) The disambiguation of polysemantic words, phrases and sentences
(h) Discourse analysis of paragraphs and larger units of natural language text
(i) The failure to incorporate the notions of model, extension, and intention and the provision to account for decoding
(j) All the systems described, when looked at individually, fall short of the requirements for a semantic theory that is adequate for natural languages

There is one thing which is more or less common to approaches which have been described: the structure which represents the relevant conceptual content of a natural language string must be a formal (artificial) language with the power of a higher order logic. According to Simmons[42] variables and operators in this formal language can be represented as nodes and labeled connections which can be taken as a model of human cognitive structures. Such a formal language representation of the natural language content corresponds to the semantically interpreted deep structure of Chomsky's transformational theory.

The generalization of natural language statements is achieved by the application of a set of transformations to constituents of statements of formal language to produce the surface structure. However, the problem of the control of coherence and connectivity between statements is still unresolved.

We assume that the underlying cognitive structures of people who speak a certain language are structurally similar and can be represented by the same formal language. Consequently, the development of a formal intermediate language could serve as a basis for machine translation of a source language into intermediate language and from intermediate language into several target languages.[40]

A computer-oriented formalization of natural language semantics will have a variety of practical applications such as machine translation, development of query-answering systems, automated abstracting and indexing of documents and others. It can be expected that the use of the computer to develop models of human cognitive processes will provide insights into the nature of a semantic theory.

Developments in the past five years in the area of semantically-oriented automated language processing systems have been reviewed independently by Simmons[41,42] Bobrow, Frasier and Quillian,[43] Salton[44] and Montgomery.[45]

III. Conclusions

Most semantic theories and semantically oriented computer systems are based on models of human cognitive processes. Even K-F semantic theory can be included in this group.

The basic elements of cognitive processes are concepts--we call them categorial semantic units which have the function of semantic nuclei, and which are co-associated by a set of semantically distinct relational predicates. In the correlation grammar the concepts are called correlata and relational predicates correlators, in Quillian's semantic network units and properties consisting of attribute-value pairs; in K-F theory they are denoted as semantic markers, distinguishers and projection rules, in Pratt's semantic model for medical English they are called semantic categories and relational predicates; in some other approaches they are referred to as one-referent and two-referent predicators.[46]

If a semantic theory is suitable for computation, it must be translated into a formal language with the power of higher order logic including logical inferences.

Very significant weaknesses are still apparent. All existing systems and/or models are experimental, small and core bound. None uses more than a few hundred words of dictionary, with the exception of the SNOP lexicon, a small grammar and a restrictive semantic system. No system is able to deal with more than a small subset of English and, in general, they lack adequate heuristic controls to limit the extent of searching an infinite inference tree. Little has been done so far to incorporate inductive inference procedures. Very few systems have tried to go beyond sentence boundaries in their analysis (Olney[47]) and attack the problem of anaphoric and elliptical constructions. There exists no operational systems for the semantic interpretation of even a relatively restricted set of objects, associations, and functions, and there exist no satisfactory deductive system that would make it possible to generate even single inferences in a resonable amount of time. The number of admissible functions and associations is restricted and the class of semantically well defined objects is rather small. The few existing systems such as Deacon, TLC, SIR and others are based on the manipulation of semiformal input languages for which the semantic problem is either eliminated or drastically reduced.

In general, all the mentioned theoretical proposals are highly tentative. The definitions of semantic processes and dictionary definitions should be formulated more rigorously. This is partially due to the fact that it is unknown whether a unique, consistent hierarchization of the semantic features of language is possible.

Semantics is the least understood, most difficult, and most important area for further research and development in information retrieval language design. Work in formal and empirical semantics and semantic studies of natural languages must be continued.

What can we expect to be done in the future? There is a need in semantics for fresh empirical evidence obtained by studies of concrete lexical data. It will be necessary to prepare large dictionaries and correspondingly large, generative grammars and semantic systems based on formal theories of semantics. This will introduce a whole new order of complexity and may require the discovery of entirely new techniques to accomplish the same goal. It is our belief that the development of a sophisticated system for automated processing of natural language is a realistic goal even if it is not possible to predict when this goal will be achieved.

"Difficulties begin with man" and not always with the computer, and we hope that man will be able to resolve these problems which are primarily his.

REFERENCES

(1) WITTGENSTEIN,L. Philosophical Investigations. The MacMillan Co., 1953.

(2) ALSTON,W.P. Philosophy of Language. Prentice-Hall, Inc., 1964.

(3) CHAPIN, P.G. A Procedure for Morphological Analysis. Information System Language Studies, No. 18, The Mitre Corp., Bedford, Mass., 1968.

(4) EARL, L.L. Automatic Determination of Parts of Speech of English Words. Mechanical Translation and Computational Linguistics, Vol. 10, No. 3, 4, The University of Chicago Press, 1967.

(5) PACAK,M. Computational Morphology. Papers in Linguistics in Honor of L. Dostert, Ed. by W. Austin, Mouton Co., The Hague, 1967.

(6) PRATT, A.; PACAK, M. Identification and Transformation of Terminal Morphemes in Medical English. Methods of Information in Medicine , Vol. 8, No. 2, F.K. Schaltauer Verlag, Stuttgart, 1969.

(7) BOBROW, D. G. Syntactic Theory in Computer Implementations. Automated Language Processing, Ed. H. Borko, John Wiley and Sons, 1968.

(8) KUNO, S. Automatic Syntactic Analysis. Seminar on Computational Linguistics, Ed. A. W. Pratt, A. H. Roberts, Public Health Service Publications, No. 1716, U.S. Government Printing Office, 1967.

(9) HAYS, D. G. Dependency Theory: A formalization and Some Observations. Language, Vol. 40, No. 9, 1964

(10) BAR-HILLEL, Y. The Outlook for Computational Semantics. Proceedings of the Conference on Computer-related Semantic Analysis, Clearinghouse for Federal and Scientific Information, AD-636-508.

(11) CHOMSKY, N. Aspects of the Theory of Syntax. M.I.T. Press, Cambridge, Mass., 1965.

(12) KATZ, J. J. Recent Issues in Semantic Theory. Foundations of Language, 1967.

(13) KATZ, J. J.; FODOR, J. A. The Structure of a Semantic Theory. Readings in the Philosophy of Language, Prentice-Hall, Inc. Englewood Cliffs, New Jersey. 1965.

(14) HARRIS, Z. S. Structural Languistics. The University of Chicago Press, 1961.
(15) BOLLINGER, D. The Atomization of Meaning. Language, Vol 41, No. 4, 1965.
(16) WEINREICH, U. Explorations in Semantic Theory. Current Trends in Linguistics, Ed. T. A. Sebeok, Mouton Co., The Hague, 1966.
(17) THORNE, J. P. A Program for the Syntactic Analysis of English Sentences. Communications of the ACM, Vol. 12, No. 8, August 1969.
(18) GARVIN, P. A Linguist's View of Language Data Processing. Natural Language and the Computer, Ed. P. Garvin, McGraw-Hill, 1963.
(19) GARVIN, P. Predication Typing: A Pilot Study in Semantic Analysis. Language, Supplement 43 (2), Pt. II, 1967.
(20) FILLMORE, CH. J. Toward a Modern Theory of Case. The Ohio State University, R. F. Project 1685, Report No. 13, Columbus, Ohio.
(21) FILLMORE, CH. J. The Case for Case. Universals in Linguistic Theory, E. Bach and R. Harmus (Eds.), Hold Rinehard and Winston New York, 1968.
(22) CECCATO, S. et al. Mechanical Translation, The Correlational Solution. Clearinghouse for Federal Scientific and Technical Information, AD-409607.
(23) SCHWARCZ, R. M. Towards a Computational Formalization of Natural Language Semantic. International Conference on Computational Linguisticas, Preprint No. 29, Stockholm, Sweden, September 1969.
(24) SCHWARCZ, R. M. Steps Toward a Model of Linguistic Performance. Mechanical Translation and Computational Linguistics, No. 10, 1967.
(25) BENJAMIN, A. C. Operationism. Charles C. Thomas, Springfield, Inc., 1955.
(26) QUILLIAN, R. Semantic Memory. Semantic Information Processing, Ed. M. Minsky, Cambridge, M.I.T. Press, 1968.
(27) QUILLIAN, R. The Teachable Language Comprehender. Communication of the ACM. Vol. 12, No. 8, August 1969.
(28) ABELSON, R. P.; CAROLL, C. M. Implicational Molecules, A Method for Extracting Meaning from Input Sentences. Proceeding of International Conference on Artificial Intelligence, Washington, D. C., 1969.
(29) COLBY, K. M.; TESLER, L. and ENEA, H. Experiments with a Search Algorithm for the Data Base of a Human Belief System. Proceedings of International Conference on Artificial Intelligence, Washington, D. C., 1969.
(30) KELLOGG, CH. H. A Natural Language Compiler for On-line Data Management. AFIPS Conference Proceedings, Vol. 33, Thompson Book Co., Washington, D. C., 1968, pp. 473-493.
(31) SIMMONS, R. F. Linguistic Analysis of Constructed Responses in CAI. University Texas, Computation Center, TNN-86, 1968.
(32) SIMMONS, R. F.; BURGER, J. F. and SCHWARCZ R. M. A Computational Model of Verbal Understanding. AFIPS Conference Proceedings, Vol. 33, Thompson Book Co., Washington, D. C., 1968.
(33) WOODS, W. A. Semantic Interpretation of English Questions on a Structured Data Base. Mathematic Linguistics and Automatic Translation, Report NSF 19, 1967, The Computational Laboratory, Harvard University, Cambridge, Massachusetts.
(34) WOODS, W. A. Procedural Semantics for a Question-Answering Machine. Proceedings AFIPS, Fall 1969, Thompson Book Co., Washington, D. C.
(35) PRATT, A. W.; PACAK, M. G. Automatic Processing of Medical English. International Conference on Computational Linguistics, Stockholm, Sweden, September 1969.
(36) Systematized Nomenclature of Pathology. College of American Pathologists, 1965.
(37) MONTGOMERY, CH. Linguistics and Automated Language Processing. Proceedings of International Conference on Computational Linguistics, Stockholm, Sweden, Preprint No. 41, September 1969.
(38) PACAK, M. G. Homographs I: Classification and Identification. Vistas in Information Handling, Ed. P. Howerton, Spartan Books, Washington, D. C., 1963.
(39) PACAK, M. G. Homographs II: Classification and Identification. Etudes de Linguistique Appliques, No. 5, Didier Publ., Paris, 1967.
(40) OTTEN, M; PACAK, M. G. Intermediate Languages for Automatic Languages Processing. Third International Symposium on Computer and Information Sciences, Bal Harbour, Florida, December 1969.
(41) SIMMONS, R. F. Answering English Questions by Computer. in Automated Language Processing, Ed. H. Borko, John Wiley and Sons, New York, 1967.
(42) SIMMONS, R. F. Natural Language Question Answering Systems. Communications of the ACM, Vol. 13, No. 1, January 1970.
(43) BOBROW, D. G.; FRASER, J. R. ; QUILLIAN, M. R. Automated Language Processing. Review of Information Science and Technology, Ed. C. A. Cuadra, Vol. 2, Encyclopedia Brittanica, Chicago, 1967.
(44) SAlTON, G. Automated Language Processing. Review of Information Science and Technology, Ed. C. A. Cuadra, Vol. 3, 1968.
(45) MONTGOMERY, Ch. Automated Language Processing. Review of Information Science and Tachnology, Ed. C. A. Cuadra, Vol. 4, Encyclopedia Brittanica, Chicago, 1969.
(46) HOCKETT, CH. F. The Problem of Universals in Language. Universals of Language, Ed. J. H. Greenberg, The M.I.T. Press, Cambridge, Massachusetts, 1966.
(47) OLNEY, J.; REVARD, C. and ZIFF, P. Toward the Development of Computational Aids for Obtaining a Formal Description of English. System Development Corporation, SP-2766/001/00, October 1968.
(48) LAMB, S. An Outline of Stratificational Grammar. Georgetown University Press, Washington, D. C., 1966.
(49) von GLASERFELT, E.; PISONI, P. The Multi-

store Parser for Hierarchical Syntactic Structures. Communications of the ACM, Vol. 13, No. 2, February 1970.

(50) STOCKWELL, R. P.; SCHACHTER P.; PARTEE, B. H. Integration of Transformational Theories on English Syntax. Vols. I, II, ESD-TR-68-419, UCLA, October, 1968.

(51) CHOMSKY, N. Remarks on Nominalization. to appear in Jakobs and Rosenbam (Editors) "Readings in English Transformational Grammar.)

(52) MOYNE, J. A. A Progress Report on the Use of English in Information Retrieval. Proceedings of the International Conference on Computational Linguistics, Stockholm, 1969.

(53) MOYNE, J. A. Information Retrieval and Natural Language. Proceedings of the American Society for Information Science, Vol. 6, 1969.

HOW FEATURES RESOLVE SYNTACTIC AMBIGUITY

Bozena H. Dostert and Frederick B. Thompson
California Institute of Technology, Pasadena, California

ABSTRACT

Ambiguity is a pervasive and important aspect of natural language. Ambiguities, which are disambiguated by context, contribute powerfully to the expressiveness of natural language as compared to formal languages. In computational systems using natural language, problems of properly controlling ambiguity are particularly large, partially because of the necessity to circumvent parsings due to multiple orderings in the application of rules.

Features, that is, subcategorizations of parts-of-speech, constitute an effective means for controlling syntactic ambiguity through ordering the hierarchical organization of syntactic constituents. This is the solution adopted for controlling ambiguity in REL English, which is part of the REL (Rapidly Extensible Language) System. REL is a total software system for facilitating man/machine communications. The efficiency of processing natural language in REL English is achieved both by the detailed syntactic aspects which are incorporated into the REL English grammar, and by means of the particular implementation for processing features in the parsing algorithm.

KEY WORDS AND PHRASES

REL System, natural language, syntax, grammar, semantics, features, parsing, computational linguistics, man/machine communication, ambiguity

HOW FEATURES RESOLVE SYNTACTIC AMBIGUITY

I. Introduction

Natural language, e.g. English, is highly ambiguous. Computational analysis of natural language has added a new dimension to the discovery of language ambiguity, and, not infrequently, created problems in ambiguity which are not transparent to the ordinary linguist, and even less to the ordinary language user.

Many sentences may have more than one interpretation, for example: "The students suggested that they might stop smoking in the meetings." Even more commonly, phrases that occur as parts of sentences may be highly ambiguous. For example: "German books" may mean (a) books in German, (b) books on German, (c) books made in Germany, and so on. Ambiguity, either at the phrase level or at the sentence level, is a powerful mechanism in language. Ambiguity inherent in syntax allows for fewer syntactic constructions to be used in expressing meanings. If every phrase and sentence of the language had to be unambiguous in isolation as well as context, the amount of syntactic structure to render different meanings would have to be considerably increased. Ambiguities, which are disambiguated by context, contribute powerfully to the expressiveness of natural language as compared to formal languages.

This paper is concerned with ambiguity problems as they occur in the environment of a computational system which allows the use of natural language in man/machine interaction. Syntactic ambiguity inherent in natural language is only part of the problem that faces a linguist in this environment. He must also deal with ambiguity which is, in a sense, his own creation, or more precisely, the result of the simple-mindedness of the algorithms that he is forced to impose on the analysis of language. Thus:

Where was the brother of John's wife in 1966?

might be intuitively unambiguous. Yet from the point of view of the parsing algorithm these interpretations are possible:

Where was the (brother of ((John's wife) in 1966)?

Where was the (brother of (John's wife)) in 1966)?

Where was the ((brother of John)'s wife) in 1966)?

All overt clues present in a language must be utilized for disambiguation, and ways must be found to circumvent multiple parsings where no inherent ambiguity exists.

In the case of systems working with a data base (some well defined universe of discourse) ambiguity can often be resolved by reference to the data base -- as indeed it is commonly resolved in natural language. It has been found however that it is often time-saving to complete syntactic analysis of a sentence before proceeding to establishing its semantic interpretation. All possible syntactic analyses of a sentence must therefore be considered. Further, where only one interpretation is possible, and only the order of constituent analysis is to be determined, the choice of parsing can greatly influence the subsequent semantic computations. Thus, in the case of:

John's (son who lives in Boston)

(John's son) who lives in Boston

the former parsing will require far more accesses to the data base, since all ambiguous readings of "son who lives in Boston" have to be found. To put it another way, John is likely to have fewer sons than there are sons who live in Boston.

Resolution and control of syntactic ambiguity requires attention to the overt syntactic clues present in the input sentence. These may be morphological, such as inflectional suffixes, agreement in gender, number and case, clues provided by the syntactic environment such as determiners and auxiliary verbs, and positional clues such as inversion of subject and verb in questions.

One way of exploiting some of these clues is to use multiple parts-of-speech. Thus one might have both <possessive noun>and <nominative noun>, instead of just <noun>, to distinguish between "boy's" and "boy." This, however, leads to further subcategorizations, and thus rapid proliferation of parts-of-speech: <possessive plural noun>and <possessive singular noun> for "boys'" and "boy's." An immediate undesirable result of this solution is the proliferation of the rules of grammar. Thus, a simple rule such as:

<noun> → <article> <noun>

must be replaced by several, e.g.:

<determiner modified nominative plural noun>→

<article> <non-determiner modified nominative plural noun>.

A second device, one that is often resorted to, is to adopt special parts-of-speech whose only function is to make explicit such syntactic clues. Thus one might have the context-sensitive rule:

<noun> <possessive> → <noun>'s

There again problems arise. To parse "boys' location" and "boy's location," we would need such rules as

<noun> → <noun> <plural> <possessive> <noun>

<noun> → <noun> <singular> <possessive> <noun>

The result is again a great multiplying effect on the rules of grammar and on parsing time.

A major defect of both of the above solutions is that they increase markedly, rather than decrease, the syntactic ambiguity they seek to control.

The device which we have employed to control syntactic ambiguity is features. The role of features is dual: subcategorization of parts-of-speech and the determination of the order of constituent analysis. As an example of the former, the word "boys'" is assigned the simple part-of-speech <noun> and the plural and possessive features. As an example of the latter, in the case of "(brother of ((John's wife) in 1966," parsing proceeds in two steps: (1) the normal matching of a grammar rule to a segment of the parsing graph; (2) a check to determine whether the features assigned to the constituent phrases match those prescribed by the rule. The explicit mechanisms are discussed in Section IV. As will be seen, the algorithm we have developed is indeed efficient. In Section III, we present the features that we are now using and examples of their application in syntactic disambiguation. First, however, we need to characterize the particular computational linguistic system we are using, namely REL.

II. The REL Environment

The REL (Rapidly Extensible Language) System is a complete software system for communicating with the computer for the purpose of data analysis, and construction and application of conceptual models. It includes a natural component, REL-English, which enables the user to work with the computer in a subset of natural English. [1, 2, 3]

The REL system language processor makes use of the parsing algorithm due to Martin Kay. [4] This is a bottom-to-top, right to left, single pass parser that can handle any general rewrite rule grammar. REL does not make use of a lexicon but rather takes as its terminal vocabulary all of the printable characters: upper and lower case alphabetics, digits, punctuation marks, and blank. Thus it would recognize a word such

as "John" in terms of the grammar rule:

<name> → John

The REL English parts-of-speech strike an ordinary linguist as highly unconventional. They are:

<name>	e.g. boy, tall, John
<relation>	e.g. location, sister, author
<number>	e.g. 523
<verb>	e.g. arrive, marry
<time modifier>	e.g. January, before 1960
<relative clause>	e.g. who lived in Boston

Function words (in the sense of Fries) [5] e.g. "have," "of," "and," "all," are included in rules in their literal form, for instance, "of" in the rule:

<name> → <relation> of <name>

which applies to, e.g.:

Boston → location of John

The reasons for using these rather than conventional parts-of-speech stem from the way the surface sentence is related to the corresponding data structures used for the various data bases (universes of discourse). We are aware of the fact that our parts-of-speech may make our grammar less transparent. We, in turn, find the separation between the syntactic and semantic levels of language to be an unnatural one. Our parts-of-speech reflect our view of language as a functionally integrated phenomenon. Since we are leaving the semantic part of our analysis out of account, it would not be difficult to conventionalize our parts-of-speech for the purposes of this presentation. We certainly feel that the use of features that our system exemplifies can be carried over to other systems, using conventional parts-of-speech or otherwise. Secondly, we feel it particularly important that we can cite actual operating experience, and this can be done without distortion only if we adhere to the actual form of our grammar. The REL system became operational in February 1970, and has amply demonstrated its efficiency as a time-shared relational data base question-answering system.

III. Features and Their Functions

The general notion of features as subcategorizations of parts-of-speech is certainly not new. They are implicit in a paper of G. N. Harman. [6] Chomsky discusses features at length in Aspects of the Theory of Syntax. [7] However, these and other references deal almost exclusively with linguistic issues of a somewhat different sort than we are interested in here. In particular, Chomsky's features characterize lexical items in terms of such categories, for instance, as Count, Common, Animate, Human for the entry "boy"; they also characterize the syntactic environment in which an item can occur, e.g. for "grow": + verb, which can occur with a noun phrase (grow flowers), by itself (flowers grew), and with an adjective (grow old). The primary role of our features is the ordering of the hierarchical organization of syntactic constituents with the aim of controlling syntactic ambiguity. This constitutes an interesting extension of the use of features, both from strictly linguistic and computational points of view.

Each part of speech is assigned a class of features. A particular word or phrase bearing that part of speech is checked for the presence or absence of these features.

A. Features that Structure Nominal Phrases

In nominal phrases, features are used to indicate inflectional structure (plural and possessive) and constituent structure. They mark the presence of determiners and quantifiers and enforce the desired grouping of modifiers.

The plural (PLF) and possessive (POF) features result from the following rules:

$$\text{<name>}_{1+PLF} \rightarrow \text{<name>}_{-PLF}\ s$$

e.g. boys → boy s

This rule allows the plural "s" to go on a name which is not plural (-PLF) and results in a name that is marked as plural (+PLF). The "1" means that the features of the first constituent on the right-hand side are also carried over and assigned to the resulting left-hand side.

The rules for possessive, singular and plural, are:

$$\text{<name>}_{1+POF} \rightarrow \text{<name>}_{-PLF-POF}\ 's$$

e.g. boy's

$$\text{<name>}_{1+POF} \rightarrow \text{<name>}_{+PLF-POF}\ '$$

e.g. boys'

The possessive is set last, after all premodification, as controlled by other nominal features

e.g. (big dog)'s master.

The determiner (DTF) and quantifier (QNF) features result from these rules:

$$\text{<name>}_{1+DTF} \rightarrow \begin{bmatrix} the \\ a \end{bmatrix}\ \text{<name>}_{-DTF}$$

e.g. the boy

By requiring that the right-hand side <name> must not be modified by a determiner (i.e. the DTF feature must be off), the rule prevents such ungrammatical sequences as

* the the boy

Examples of quantifier rules are:

$$\langle name\rangle_{1+QNF-PLF} \rightarrow \text{some } \langle name\rangle_{-QNF-DTF}$$

$$\rightarrow \text{all } \langle name\rangle_{-QNF+PLF}$$

$$\rightarrow \text{all of } \langle name\rangle_{-QNF+PLF+DTF}$$

These rules allow such sequences as exemplified above plus such as "some boys" (the plural feature is not marked in the rule), "all the boys" (DTF is not marked in the rule) and rule out such ungrammatical sequences as * some the boy (DTF must be off) and * all boy (PLF must be on).

The setting of determiner and quantifier features prevents further adjectival or possessive modification. The remaining features govern the order of constituent analyses (in conjunction, of course, with the ones above). This is necessitated by the fact that noun phrases are hierarchically structured, i.e. some constituents serve as modifiers of others, and the ordering of the sequence of modification determines the meaning of a given phrase. Some phrases are genuinely ambiguous, e.g. "Jane's children's book" can mean either 'the book of Jane's children' or 'a children's book which is in some relationship with Jane, e.g. owned by her.' Other examples are the familiar "stout major's wife" or "little boys' school." But in computational analysis, phrases which are normally unambiguous also turn out to have alternate analysis, thus "wealthy benefactor's statue" parses both ways, i.e. "(wealthy benefactor)'s statue" and "wealthy (benefactor's statue)." The second is obviously incorrect, and our features are directed at discarding such analyses.

We distinguish between (1) adjectival (APF), (2) possessive (PSF), (3) prepositional ("of"-phrase) (PMF), and (4) relative clause modification (RCF), e.g. (1) old uncle, (2) John's uncle, (3) uncle of John, (4) uncle who attended Harvard.

APF marks phrases resulting from modification by nouns other than possessive and adjectives.

PSF marks phrases resulting from modification by a possessive noun.

PMF marks phrases containing "of."

RCF is discussed in greater detail below.

The following five rules illustrate rules used to control groupings.

(1) $\langle name\rangle_{2+APF} \rightarrow \langle name\rangle_{-APF-DTF-PMF-POF-PSF-RCF-QNF}$

$\langle name\rangle_{-DTF-PMF-POF-PSF-RCF-QNF}$

Yale women

(2) $\langle name\rangle_{2+PSF} \rightarrow \langle name\rangle_{-DTF-PMF+POF-RCF}$

$\langle name\rangle_{-DTF-PMF-POF-PSF-RCF-QNF}$

Scott's Ivanhoe

(3) $\langle name\rangle_{2+PSF} \rightarrow \langle name\rangle_{-DTF-PMF+POF-RCF}$

$\langle relation\rangle_{-DTF-PMF-POF-PSF-RCF}$

John's sister

(4) $\langle name\rangle_{1+PMF} \rightarrow \langle relation\rangle_{-DTF-POF-PSF-RCF-PMF}$

of $\langle name\rangle_{-POF}$

location of John

(5) $\langle name\rangle_{1+RCF} \rightarrow \langle name\rangle_{-DTF-PMF-POF-RCF}$

who <verb>

women who left Yale

Adjectival structures are grouped first, i.e. before possessive and "of"-phrases,

e.g.: John's (old uncle)

: (crowded New York)'s subways

: dormitories of (Yale women)

: sister of (uncle John)

If there is more than one adjective, they group to the right,

e.g. good (old uncle).

Possessive structures are formed after adjectival structures and before "of"-phrases are put on,

e.g. friend of (Chicago's mayor).

If there are more than one possessive, they group to the left,

e.g. (John's son)'s teacher.

If there are more than one "of"-phrases, they group to the right,

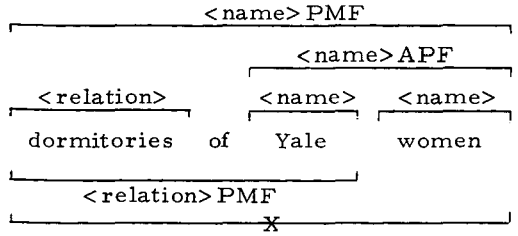
e.g. sister of (the father of John).

Modification by a pre-posed possessive and a post-posed "of"-phrase is prevented,

e.g. * New York's governor of New York

* John's sister of Boston

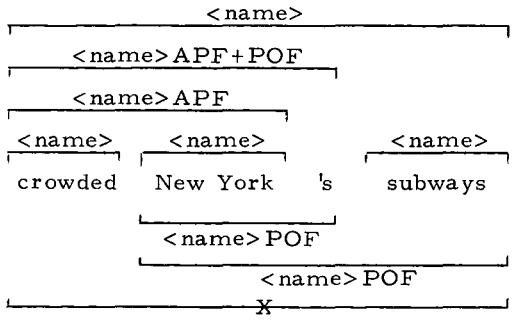
How are these results achieved and is the control of ambiguity according to our solutions desirable in all cases? Let us consider some examples of our analyses.



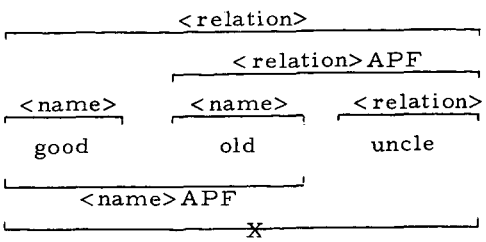
The bottom parsing is prevented by PMF since the rule

<name> → <relation> <name>

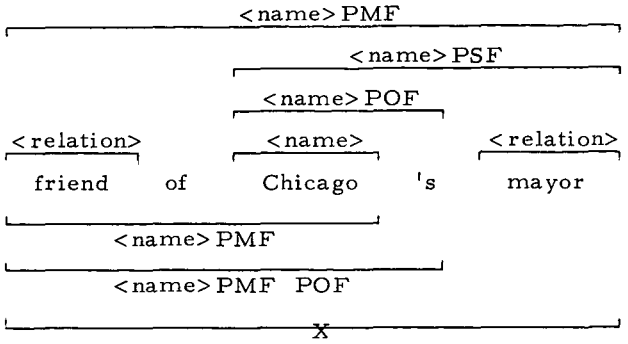
requires the PMF (prepositional modification) to be off on the <relation>. The correct analysis is achieved here.



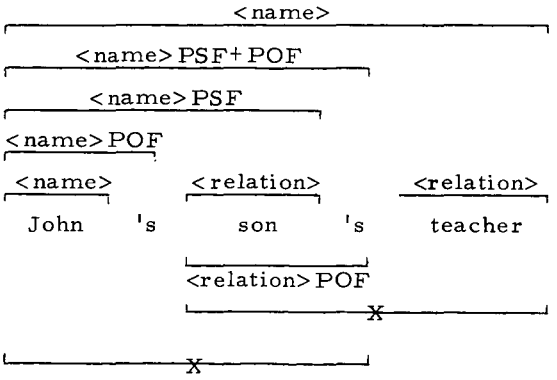
The bottom parsing is prevented, since rule (1) combining <name> and <name> calls for the possessive PSF to be off on the second name. But this is clearly a case of over-disambiguation, since both readings are possible. Correct disambiguation is obtained in "(wealthy benefactor)'s statue," but incorrect in "(wealthy building)'s proprietor."



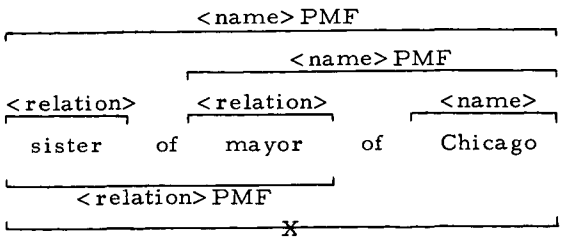
The rule combining <name> and <relation> requires that the <name> be not modified by an adjective (APF) and thus the rule does not apply. But in the case of "Vermont (marble statues)" or "light (grey suit)" we would have over disambiguation.



The rule combing <name> with <relation> requires that the PMF must be off on the <name>. However, we also exclude "(mayor of Chicago)'s sister."



The rule combining <relation> with <relation> calls for the POF to be off on the first <relation> and the rule combining <name> and <relation> calls for the POF to be off on the <relation>. These rules exclude, however, * "Jill's (children's book)" which is legitimately ambiguous.



The rule combining <relation> of <name> requires the PMF to be off on the <relation>.

The remaining feature, RCF, marks phrases consisting of nouns modified by relative clauses. Such modification is subsequent to adjectival and possessive modification, e.g. "(old man) who left Boston," "(John's friend) who left Boston," and precedes "of"-phrase modification, e.g. sister of (the boy who left Boston).

An interesting constraint on constructions involving relative clauses is that the noun on the left-hand side of "of" cannot be modified by a relative clause unless the noun on the right-hand side is also, thus

* the sister who left Boston of John

is ungrammatical, while

the sister who attends Yale of the boy who attends Harvard

is well-formed. The ungrammatical construction is prevented by rules which require parallel occurrence of the RCF feature on both nouns.

Another interesting use of features and the comma is the resolution of ambiguity in constructions involving relative clauses such as

sister of the boy who left Harvard

in which the relative clause may refer to either of the nouns. There are two sets of rules, with and without the comma, e.g.

<name> → <name> who <verb>

<name> → <name>, who <verb>

and together with the function of features they result in different groupings:

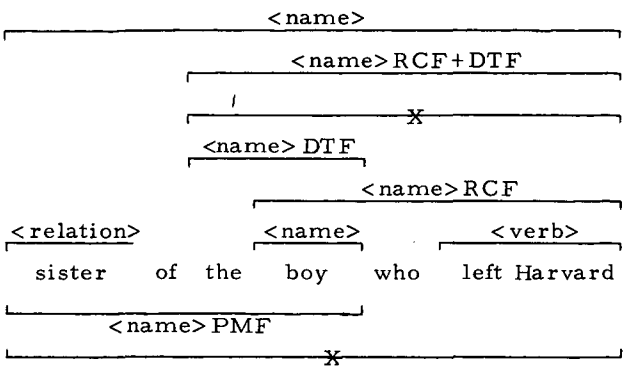
sister of (the boy who left Harvard)

(sister of the boy), who left Harvard

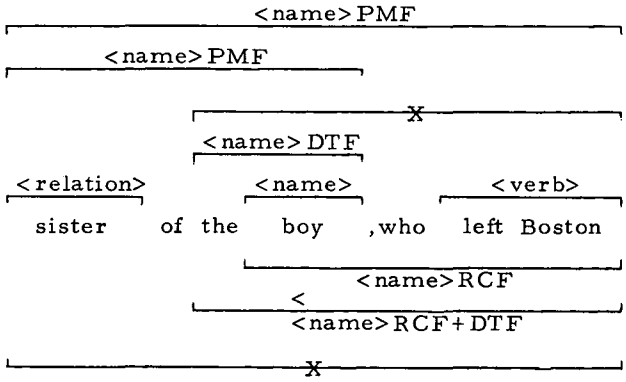
boys who knew (girls who left Boston)

(boys who knew girls), who left Boston.

How this is achieved can be observed in the following examples:



The bottom parsing is discarded since the rule which puts the relative clause on the <name> requires that the <name> must not be modified by an "of"-phrase. Also, the parsing combining "the boy" with "who <verb>" is prevented by requiring that the <name> must not be determiner modified.



The bottom parsing is excluded since the rule combining <relation> and <name> prohibits the name to be modified by a relative clause.

As we have seen, features are a powerful mechanism in controlling undesirable ambiguity. In some cases, their use leads to over-disambiguation. This could be avoided by relaxing their functions, and allowing ambiguous parsings to show up. Unfortunately, undesirable and illegitimate ambiguities would also show up. Our purpose is to facilitate man/machine communication rather than exhaustive linguistic analysis, and thus we have strived to find a balance between computing efficiency and freedom of language use. We have not reached a final conclusion in some of these matters and may relax certain of our feature rules to allow more ambiguous parsings when their disambiguation is likely to result from semantic checks. But final decisions must await more experience with the system.

B. Features that Structure Verb Phrases

One essential departure in our terminology, which reflects our grammatical analysis, is that a verb phrase is not limited to verb + noun phrase (+ modifiers), but includes, in the final stages of analysis, also the subject.

Limitations of space preclude detailed discussion of verb phrase features, and only a few interesting examples are considered.

Verb phrase features mark:

(1) Inflectional shape of verbs:

FSI - singular, which restricts the verb to singular subjects. The rules which call for it involve: "does, doesn't, begins, ceases," "is, was, has," and the third person singular marker "s," e.g. "arrives."

FPP - past participle, in the usual sense. Modification by "be" and "have" auxiliaries is permitted, but no other. This form is an allowed verbal modifier of nouns, e.g. "married man." The feature is set by "d" (or "ed") morphemes and "had," "been" and "begun to."

(2) Negative modification:

FNG - set by "not" and auxiliaries containing "not," e.g. "won't."

(3) Copula verbs:

FCV and FHV mark, respectively, "be" and "have." FCV is the encompassing feature to deal with copula verbs in general, which results in fewer rules.

(4) The following features determine the order of parsing:

FPH - phrasal, FLC - left collecting, and FTM - tense modifier. Examples are:

for FPH:

(John (enters)) vs. * (John (enter)s)

i.e. when a phrase is built, the FPH prevents putting the morphemic modification "s" on the phrase rather than on the verb itself.

for FLC:

(John enters) Harvard vs. *John (enters Harvard)

((Is (John seen)) by Mary?

* Is ((John seen) by Mary)?

* ((Is John) seen) by Mary?

i.e. the subject and auxiliaries are combined with the verb before the object is put on.

for FTM:

(lived in Boston) (from 1960 to 1965)

* (lived in Boston from 1960) to 1965

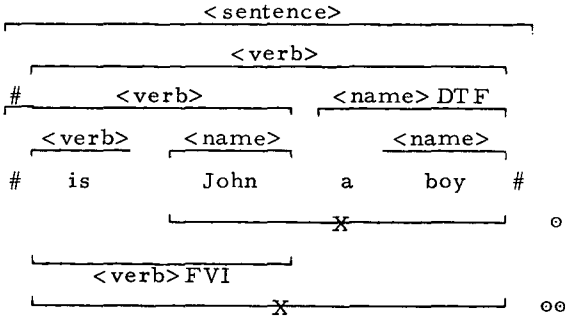
(5) The remaining three features are used to deal with deep structural relations in verb phrases.

(a) FPA, passive, is required by the rule which puts the surface object, preceded by "by," as deep structure subject. The feature is turned on by the "be" auxiliary on transitive verbs, which are distinguished from intransitive verbs by the feature FVI. An example is "the dog is owned by John," where "John" is the deep subject and "dog" is the deep object. Ungrammatical strings such as * "the dog owns by John" and * "the dog is owned John" are not permitted by the rules. The specific rules are (only relevant features are shown):

$$\langle \text{verb} \rangle \rightarrow \langle \text{verb} \rangle_{-\text{FPA}-\text{FVI}-\text{FHV}} \quad \langle \text{name} \rangle_{-\text{POF}}$$

$$\langle \text{verb} \rangle \rightarrow \langle \text{verb} \rangle_{+\text{FPA}} \text{ by } \langle \text{name} \rangle_{-\text{POF}}$$

(b) FVI, intransitive verb. One of its important roles is to allow the "be" auxiliary without triggering the passive transformation, e.g. "born" can be defined as "begun to be alive" and such phrases as "was born" are not treated as passives. Another function is to reject an incorrect parsing in questions beginning with "be" or "have."



⊙ (fails because of determiner)
⊙⊙ (fails because of FVI)

The final rules are (features omitted)

<verb> → # <verb> <name>

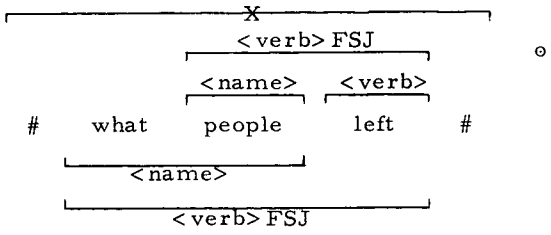
<sentence> → # <verb> #

(# is the boundary symbol). By marking "is John" as intransitive, the feature does not allow "a boy" to go on as object, forces the recognition of "John" as subject, and forces the rule which utilizes the boundary symbol to give the correct analysis.

(c) The last feature, FSJ, marks the presence of the syntactic subject. It disambiguates such sentences as

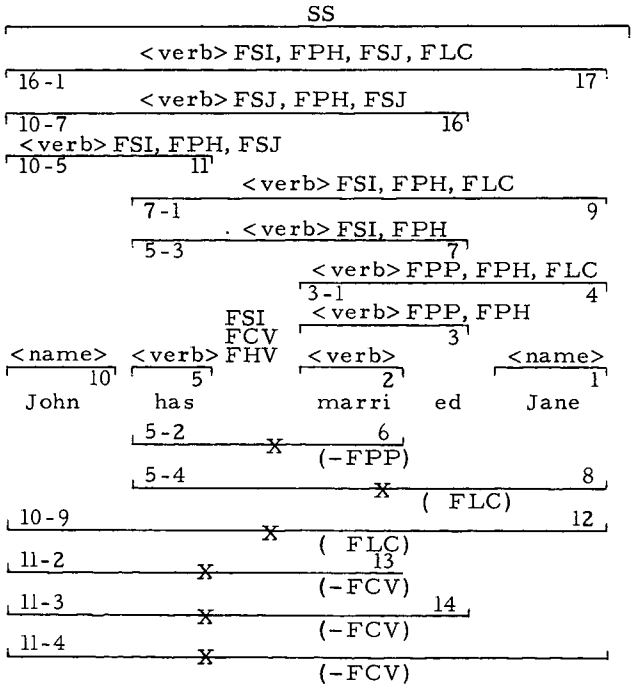
What people left?

(the second interpretation being "What did people leave?"). Only the first interpretation is allowed. The analysis is as follows (as usual, only relevant features are shown).



⊙ (fails because of FSJ)

The function of features can best be observed by following the parsing of a sentence:



Let us follow the top parsings:

Parsing procedes from right to left. Thus, "Jane" is parsed into a <name> first. Next, "marri" parses into a <verb>. Irregular verbs, like this one, are treated in a simple manner: they are regularized by defining, i.e. "marri" is defined in terms of (as equal to) "marry." Then the past participle morpheme "ed" goes on in the usual manner. Next, "married" is combined with the object "Jane" by the rule that puts objects on verbs. The feature of 'left collecting' (FLC) is then set to indicate that the object has been put on. The next item available for parsing is "has" and it parses into a singular copula verb. It then combines with "married" into another <verb>. That, in turn, combines with the object "Jane," again setting the 'left collecting' feature.

Next, "John" is parsed into a <name>, and combines, as subject, with "has," thus setting the subject feature (FSJ). Parsings which fail due to feature checks are discussed below. The next good parsing to take place is "John" and "has married." It carries on the appropriate features. Finally, that verb phrase is combined with the object "Jane" to give the correct analysis of the sentence.

The exclusion of incorrect parsings is a more interesting process: combining "has" with "marri" is prevented by lack of the 'participle feature' (-FPP) on "marri." Combination of "has" with "married Jane" is prevented by the 'left collecting' (+FLC) feature, which does not allow left modification of the verb once the right modification (in this case the object "Jane") has been put on. The combination of the subject "John" with "has married Jane" is prevented by the same rule. "John has" "marri" is excluded by the fact that the verb phrase "John has" is marked as not including a copula verb (i.e. "has" is treated here as no longer being a copula verb, but as a full verb, as it would be in, e.g. "John has books"). "John has" + "married" is prevented for the same reason, and so is "John has" + "married Jane." This analysis is of particular interest. It expressly resolves the ambiguity of such sentences as "John has stolen watches." In our analysis, only the reading "(John has stolen) watches" is allowed, thus ruling out

* (John has) (stolen watches)

* (John has) (married sons)

The latter two readings are allowed, though, by different rules, which limited space did not permit discussing here.

As stated at the outset, ambiguity is a basic aspect of language. However, no sentence is an

island entire in itself, to paraphrase John Donne. [8] A sentence, or a phrase, lives in its context and derives its meaning therefrom. Computational analysis is deprived of many clues, such as intonation, that resolve much of phrasal or sentential ambiguity. It must, therefore, resort to other means, and, as a payoff, it externalizes linguistic structure, which is otherwise not easily seen. It thus gives us clues to the ways language works.

The question might arise as to why we seek to control ambiguity. The answer is that computing time has to be cut down: a researcher working with the computer to obtain information would not be happy if he had to sit at the console for three minutes after having asked a question and wait while all the ambiguous parsings are being ground out. Besides, who except linguists would be interested in having several ambiguous analyses of their sentences printed out. The detailed analysis of the following sentence supplies rather interesting statistics. In the analysis of the sentence:

"What was the college that was entered by John? "

the statistics are the following: 176 rules would have applied had no feature mechanism been used; 31 rules actually applied, of which 12 were aborted by features. Of the remaining 19 parsings, only 12 were actually needed to obtain the correct analysis of the sentence. These figures speak for themselves, or, rather for the need of a controlling mechanism on ambiguities.

The computer analysis of the last sentence also pointed out an ambiguity we had originally failed to observe. Notice these sentences:

"What was the college entered by John? "

"The college that was is no more. "

Since both of these are grammatical sentences, the sentence

What was the college that was entered by John?

produced the surprising ambiguity.

In connection with a recent conference on computational linguistics [9], the comment was made that "[computational linguists] may use the computer much as economists use gold, without actually touching it. " We "touch" the computer. After all, gold is good, hard currency.

IV. The Details of Implementation

Possibly the most important aspect of the feature method is its computing efficiency. This, of course, sharply depends upon its implementation. The details of our implementation of features as a method for controlling syntactic ambiguity reflects our particular computer environment and language processor. However, once this particular implementation is seen, it can be easily carried over to other environments.

The basic parsing algorithms for handling various classes of languages are continuing to be refined. Among the best are the Martin Kay parser (a modified version of which we use in REL) [4], the Earley parser [10] and the augmented network parser of Woods [11]. However, over and beyond the basic parsing algorithm, there are two ways parsing efficiency can be lowered; first, if the number of rules that are applicable to a sentence is very large, and second, if checking on special conditions requires special subroutines. *

A large number of rules makes the parsing tree grow to great size and the time expended by even the best parsing algorithms climb. Such is the case if the number of parts-of-speech proliferates or if syntactic ambiguity, especially multiple groupings of constituents, is not controlled. These difficulties can inevitably force the computation to exceed the capacity of high speed storage; and the costs of peripheral access are prohibitive, whether it be dictionary overflow, working storage overflow, or subroutine paging. These problems are exacerbated by the small high speed memory partitions, and roll in/roll out costs of time shared systems. It is in light of these common difficulties experienced in building operational systems that we believe our implementation of features is particularly interesting.

In order to convey the computational details, we refer to the particular data structure formats we use that are associated with the parsing procedures [Fig. 1].

The dictionary uses a list structure with two-word list elements. The main portion is a binary tree holding the rules of grammar and lex-

* Parsing time is often expressed as proportional to a power of the length of the sentence being parsed, i.e. as kn^p where n is the number of words in the sentence. Parsers are compared in terms of their respective values for p. Thus, good context-free parsers are said to be proportional to n^2. However, the factor of proportionality k, which depends sharply on the implementation, is often overlooked. When disk access time, 2 to 3 orders of magnitude greater than high speed memory time, and other similar factors come into play, and since n is small in normal applications, k can be the controlling variable.

Dictionary Formats:
(2 word, 8 byte, list element)

(1) typical dictionary element

byte 1: part of speech

byte 2-4: address of definition

byte 5-6: match link

byte 7-8: fail link

(2) definition element - mask part

byte 1: code (indicating mask part)

byte 2-4: link

byte 5-6: feature on mask

byte 7-8: feature off mask

(3) definition element - second part

byte 1: code (indicating nature of definition, e.g. ambiguous)

byte 2-4: address of actual definition, including associated semantic aspects

byte 5-8: address of syntax conditions routine

Phrase Format (typical entry in parsing graph):
(3 word, 12 byte list elem.)

byte 1: part of speech

byte 2: (system use)

bytes 3-4: features

bytes 5-8: parsing graph link

bytes 9-12: link to associated semantic information, e.g. pointers into the data base.

Figure 1

ical items. Figure 2 gives a stylized illustration of such a dictionary containing just three simple rules of grammar.

Words and phrases are held in the parsing graph in phrase format [Fig. 1] using a list structure with three-word list elements. A stylized illustration of such a graph for "Bob is John's son" is shown in Figure 3. In the case of the Kay parser, the parsing graph consists of a series of nodes interconnected by labeled, directed arcs, the labels being the consituent words and phrases.

Parsing proceeds by selecting a node in the graph, say Q_i, and attempting to match rules of grammar to words and phrases immediately to the right of Q_i. The Q_i's are successively selected starting from the right and working to the left. As this matching process proceeds, phrases successfully matched are accumulated in a pushdown stack. One use of this stack is as a back track trail, so that when one path through the graph has been completely analyzed, the process can back up a step and proceed along an alternate path. When a successful match of a rule of grammar to a path segment in the graph has been found, this pushdown stack constitutes the list of constituent phrases to which the rule has applied.

Features are held in the phrase format as a 16 bit field, each bit allocated to a given feature. The bit position corresponding to a given feature thus functions as a binary switch indicating whether the feature is on or off for the particular phrase. Thus in our implementation, we allow a maximum of 16 features to each part-of-speech. We have found this tight, but adequate.

The definition found in the dictionary has three parts: (a) a stack of feature masks, (b) a syntax conditions routine, (c) the phrase which will be inserted as a labeled arc in the parsing graph as a result of successful application of the rule. The stack of feature masks contains an on-mask and an off-mask corresponding to the constituent phrases. Each of these on/off masks is 16 bits long. If, for example, a given rule specifies that the possessive feature must be on and the determiner feature must be off for the first constituent (as in the rule <name> → <name>'s <relation>), then the bit corresponding to the possessive feature is one in the on-mask and the bit corresponding to the determiner feature is one in the off-mask.

It will be recalled that we do not have a separate lexicon and that individual characters are our terminal vocabulary. Since these characters are taken directly from the user's input sentence, they never carry features. They are recognized to be characters, so one need not carry feature masks for them in the feature mask stack, thus reducing to an acceptable minimum the memory requirements [see Fig. 1].

It now becomes a straightforward and efficient procedure to read down the stack of constituent phrases and the stack of on/off masks in tandem. For each phrase, the on-mask and off-mask are respectively "and"-ed to the features for the phrase, and the results of these two tests are used to accept or reject the rules applicability in the obvious way. Since the consituent stack must be accumulated as a back track trail in any case, this matching procedure is particularly efficient, as it must be since it is one of the tight inner loops of the algorithm.

Although most rules require no further checks on applicability than those indicated above, there are rules of a more clearly transformational type, where further checks may be desired. For this purpose, the definition may contain what we refer to as a "syntax conditions routine." The syntax conditions routine, if it exists for a given rule, is called after the indicated feature mask checks have been made and prior to acceptance and insertion of the new phrase in the parsing graph. This routine may, for example, go into the parsing graph to check on structural context; it may set aside and/or check subsidiary information; it could even rearrange the parsing graph or call the parser recursively. Our direct interest here is in its ability to make more extensive feature checks. How this may function can be seen from an example.

The conjunction rules insist on parallel construction for the conjoint phrases, e.g.:

Mary's (uncle and aunt)

* (Mary's uncle) and aunt

(Mary's uncle) and (John's aunt)

* Mary's (uncle and John's aunt)

Thus the condition that must be met for the application of the conjunction rule is that certain of the features must be either on or off in the same way for both constituents. This clearly cannot be checked by the feature mask tests as applied independently to each constituent phrase. Therefore, such feature checks are embodied in the syntax conditions routines. Since one usually cannot afford the space in high speed memory to keep all such subroutines resident, access to peripheral storage is required, and therefore, computing time is increased. It is our experience that only a small fraction of the rules require such more expensive tests. In our current grammar for REL English, only 29 rules out of a total of 239 have syntax condition routines.

	part of speech	def. add.	match link	fail link
d_0:	<relation>	0	d_1	d_6
d_1:	(blank)	0	d_2	0
d_2:	0	0	d_3	0
d_3:	F	0	d_4	0
d_4:	(blank)	0	d_5	0
d_5:	<name>	$definition_1$	0	0
d_6:	<name>	0	d_7	d_{11}
d_7:	'	0	d_8	0
d_8:	s	0	d_9	0
d_9:	(blank)	0	d_{10}	0
d_{10}:	<relation>	$definition_2$	0	0
d_{11}:	(blank)	0	d_{12}	0
d_{12}:	<rel. clause>	$definition_3$	0	0

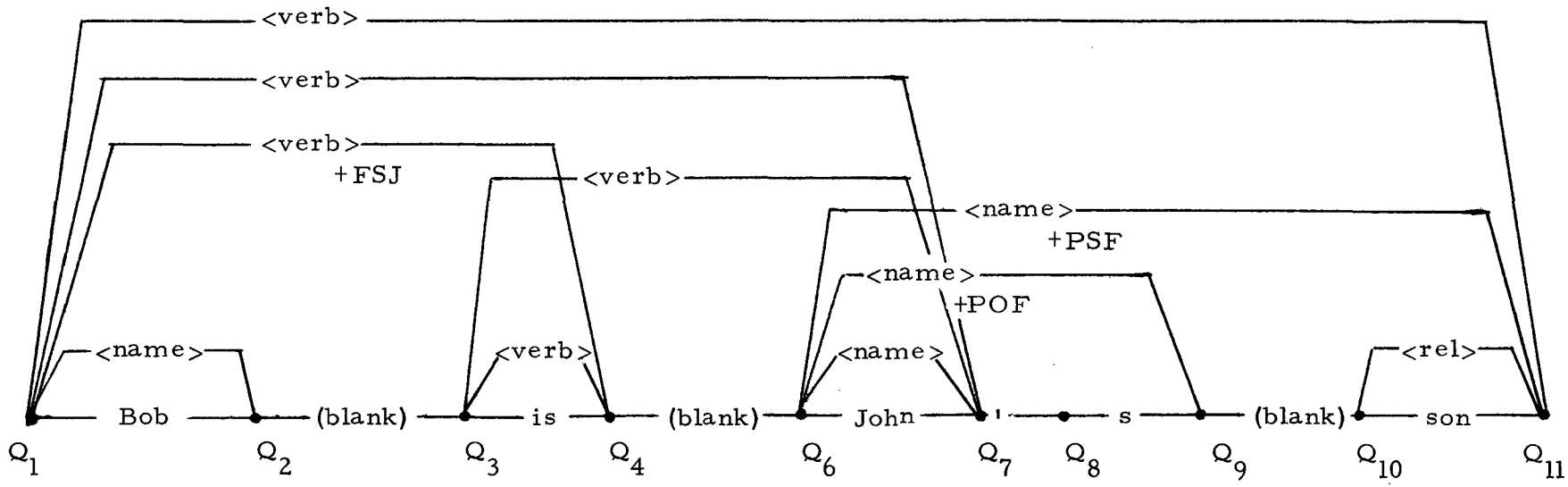
Illustration of main part of dictionary containing rules:

<name> → <relation> of <name>

<name> → <name>'s <relation>

<name> → <name> <rel. clause>

Figure 2



Typical Parsing Graph

Figure 3

REFERENCES

(1) THOMPSON, F. B.; LOCKEMANN, P. C.; DOSTERT, B. H.; et al. REL: a rapidly extensible language system. In: Association for Computing Machinery National Conference, 24th, San Francisco, Calif., August 1969. New York, 1969, p. 399-419.

(2) DOSTERT, B. H.; THOMPSON, F. B. A rapidly extensible language system: REL English. International Conference on Computational Linguistics, 3d, Stockholm, 1-4 September 1969. Preprint No. 35, 37 p.

(3) LOCKEMANN, P. C.; THOMPSON, F. B. A rapidly extensible language system: the REL language processor. International Conference on Computational Linguistics, 3d, Stockholm, 1-4 September 1969. Preprint No. 34, 31 p.

(4) KAY, MARTIN. Experiments with a powerful parser. Deuxième Conference Internationale sur le Traitement Automatique des Langues, Grenoble, 1967.

(5) FRIES, C. C. The Structure of English. Harcourt, Brace and World, New York, 1952, 304 p.

(6) HARMAN, G. H. Generative grammars without transformational rules: a defense of phrase structure. Language, 39, p. 597-616.

(7) CHOMSKY, NOAM. Aspects of the Theory of Syntax. MIT Press, Cambridge, Mass., 1965, 251 p.

(8) DONNE, JOHN. Devotions.

(9) KARLGREN, H. (Circulated letter) International Conference on Computational Linguistics, 3d, Stockholm, 1-4 September 1969.

(10) EARLEY, J. C. Generating a recognizer for a BNF grammar. Computer Science Center, Carnegie Institute of Technology, Pittsburgh, 1965.

(11) WOODS, W. A. Transition network grammars for natural language analysis. Communications of the ACM, 13 (1970) p. 591-606.

THE CONVERSE NATURAL LANGUAGE DATA MANAGEMENT SYSTEM: CURRENT STATUS AND PLANS *

Charles Kellogg, John Burger, Timothy Diller, and Kenneth Fogt
System Development Corporation, Santa Monica, California

ABSTRACT

This paper presents an overview of research in progress in which the principal aim is the achievement of more natural and expressive modes of on-line communication with complexly structured data bases. A natural-language compiler has been constructed that accepts sentences in a user-extendable English subset, produces surface and deep-structure syntactic analyses, and uses a network of concepts to construct semantic interpretations formalized as computable procedures. The procedures are evaluated by a data management system that updates, modifies, and searches data bases that can be formalized as finite models of states of affairs. The system has been designed and programmed to handle large vocabularies and large collections of facts efficiently. Plans for extending the research vehicle to interface with a deductive inference component and a voice input-output effort are briefly described.

KEY WORDS AND PHRASES

artificial intelligence, concept network, data management, deductive inference, fact retrieval, information retrieval, language processing, natural language, on-line communication, question-answering system, relational data files, semantic interpretation, syntax analysis

I. Introduction

For several years we have been engaged in research at SDC on the construction of a prototype natural-language data management system. Our general concern is with the achievement of more natural modes of communication with more complex data base structures. While present day artificial query languages may be minimally acceptable for the interrogation of simple data bases we believe that the serious compromises made in these languages between ease of learning and communication on the one hand and expressive power on the other make these languages less than adequate for coping with the more complex information files being contemplated for use on computers.

One of our principal objectives has been to develop and implement an expressive and extendable subset of ordinary English to be used to describe, modify, and interrogate data bases in a time-shared environment. The subset is expressive in that it will permit users to communicate a wide range of messages and requests to the computer in convenient grammatical patterns and extendable in that it will permit users to introduce new terms and concepts to describe a data base and create specialized English dialects especially responsive to a user's needs.

The advent of a practical system for using natural language as a means of communicating with large data bases could well lead to a much broader range of time-shared computer applications. The naive or inexperienced user should certainly welcome a means for using the familiar and convenient patterns of ordinary English in interrogating computer data files. The experienced or sophisticated on-line user may be expected to benefit as well from the ability to use the expressive power and question-answering capabilities provided by a natural-language communication facility.

A first result of investigating the feasibility of using subsets of English for communicating with structured data bases has been reported by Kellogg (6, 7). The computer program, which operated on the SDC Q-32 time-sharing system, is referred to as C-32. This program allowed an on-line user to describe, construct, and interrogate data bases of varied subject matter by use of straightforward ordinary English. However, the system was limited to small structured data bases and many types of English syntactic constructions could not be recognized or translated into computable procedures. Despite its limitations this initial program was sufficiently promising to lead us to construct a much larger and more ambitious system.

Our current program CONVERSE-360 (C-360) operates under the ADEPT time-sharing system on the IBM 360/67 computer at SDC. The principal components of this system are (1) a natural language compiler that translates English declarative, imperative, and interrogative sentences into procedural statements in a formal intermediate language, and (2) a data management system that accepts these procedural statements and carries out specified storage, search, and inferential operations.

* This work was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Information Processing Techniques Office under Contract DAHC15-67-C-0149

An overall diagram of the flow of information from user to data files is shown in figure 1. First, all words in an input sentence are looked up in the dictionary. Then, the natural language compiler employs a recognition grammar and a network of conceptual data, characterizing a universe of discourse, to apply syntactic and semantic information both to the analysis and interpretation of the input string and to the construction of procedures representing the intended message or request. These procedures, in turn, are executed by the data management system to either modify the contents of the data base or retrieve information from it.

An essential characteristic, one that provides the facilities for introducing and describing a new domain of discourse, is the ability to use existing syntactic and semantic information in the interpretation of new words. At any time, a user may introduce new nouns, adjectives, or verbs to extend the store of factual or conceptual information. When an input sentence is ambiguous or not completely interpreted, the user is alerted. He can then reformulate his request or supply additional information.

II. Research Overview

A basic premise underlying the design of C-32 was that parsing and interpretation of English sentences should yield messages and requests representable as procedures evaluatable with respect to a formal universe of discourse. This approach has been carried over and enlarged upon in the design and construction of C-360. One of the inherent complexities of natural-language processing is the need to recognize and deal with a relatively large number of abstract symbolic entities and the complex associations among them. Ten types of symbolic entities recognized and interpreted in C-360 are illustrated in figure 2. The double-headed arrows indicate that, in general, between each pair of adjacent entities many-to-many mappings may exist. It is both the number of these symbolic entities and the complexities of the relations between them that accounts for much of the difficulty encountered in the processing of natural language.

The different entities may be grouped according to their use in lexical and syntactic recognition and in semantic and conceptual interpretation. For example, lexical recognition involves the recognition of individual characters (graphemes) and their mapping to word parts (morphemes), words, and multiword terms (idioms). Each lexical item above the graphemic level is assigned one or more lists of information called lexical signatures. A lexical signature contains a specification of syntactic category and feature information as well as semantic information about one specific meaning or sense of a lexical item. In C-32 a word sense was specified in terms of a set of semantic features (see Katz, (4)) and a referent. In C-360 a word sense is characterized by a series of pointers to concepts, in a concept net, that partially define the word sense and by a referent that denotes the specific concept represented by the word sense.

Syntactic recognition and semantic interpretation are complex processes that map lexical signatures into derived signatures. The final result of sentence translation is a series of one or more derived signatures containing well-formed intermediate language procedures that represent the one or several semantic interpretations found for an input sentence. Concept net information is used during the translation process to resolve ambiguity by checking for the semantic compatibility of syntactically related constituents and to produce correct semantic interpretations.

An intermediate language procedure specifies a concept and an instruction to store, forget, find, or compute the concept. Our notion of concept is essentially due to Carnap (2). A concept is a set that may be defined in intension by means of a procedure for computing its members or in extension by enumerating its members directly. Conceptual interpretation is therefore a process of interpreting a procedure representing a concept against a data base or fact file containing information representing concepts in extensional or intensional form. Concepts may be represented in extensional or explicit form within computer memory as finite models of states of affairs--assignments of individual objects and groupings of individual objects to particular concepts. Concepts reflecting associations among individuals (functions and relations) are expressed as sets of ordered N-tuples where each N-tuple represents an elementary proposition, belief, or fact known to the system. Given these notions of concept, fact, and model we can characterize a data base description as a collection of concepts and associations among concepts in a concept network. A data base is then a collection of facts and associations among facts constituting a finite model. Put another way, a data base description or concept network is a specification of permissible or allowed states of affairs while a data base specifies an actual state of affairs. In the following sections we will show how these notions are useful in natural language analysis and question answering.

III. Translation of English into Procedures

Lexical lookup: In C-32 small dictionaries in core memory were scanned by using the hash-coding scheme built in LISP. To meet our objectives for C-360 we had to be able to handle large dictionaries (on the order of tens of thousands of words). The C-360 dictionary consists of three logical files of information stored on disc memory: a signature file, an input lexicon, and an output lexicon. The signature file contains syntactic and semantic information about lexical items. The input lexicon associates lexical items with internal names for objects, pointers to lexical signatures, and pointers to idioms in which lexical items occur. And the output lexicon is used to map internal object names into English words. The input and output lexicons are implemented as collections of LISP arrays, and LISP assembly language has been used to achieve fast access to items in an array. Each input

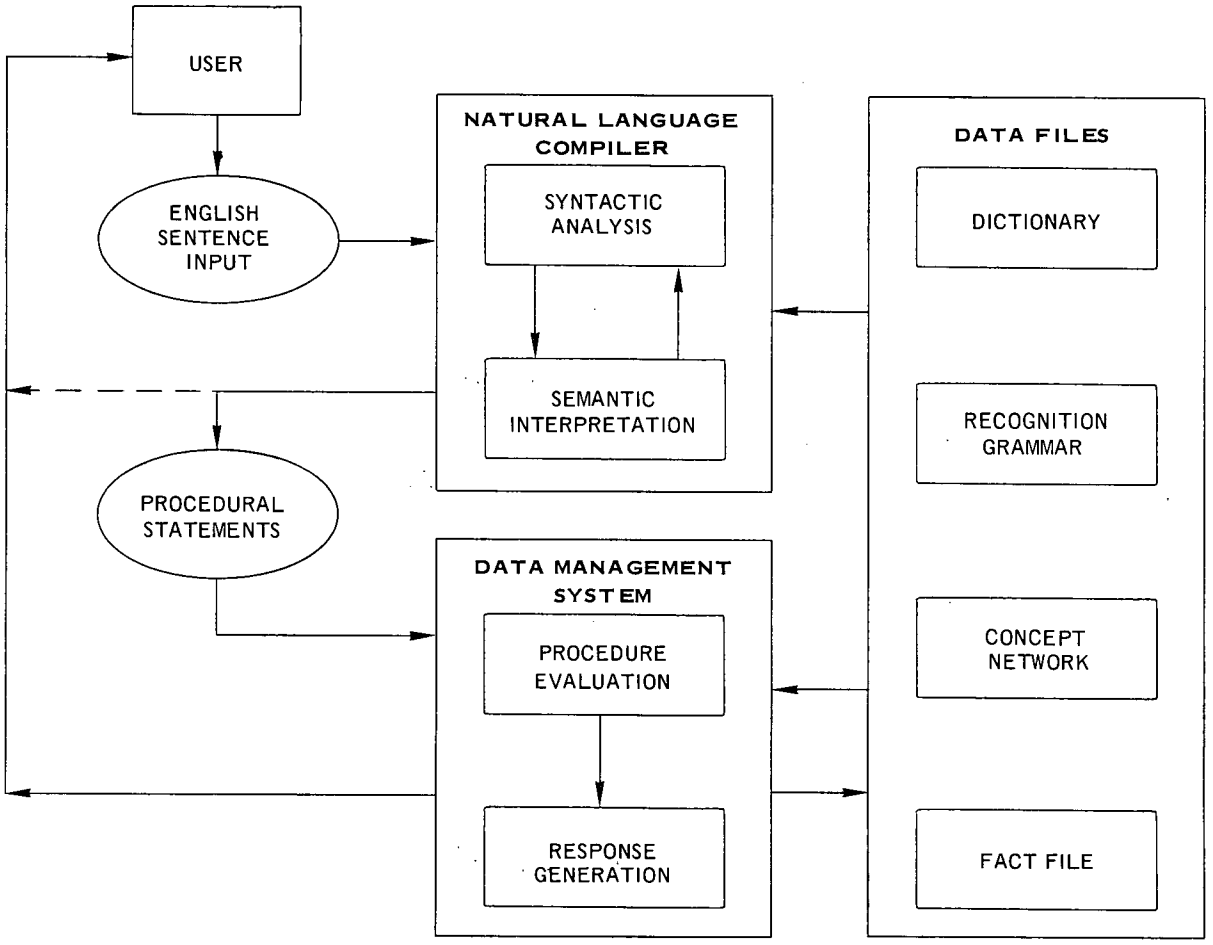


Figure 1. CONVERSE C-360 System Components

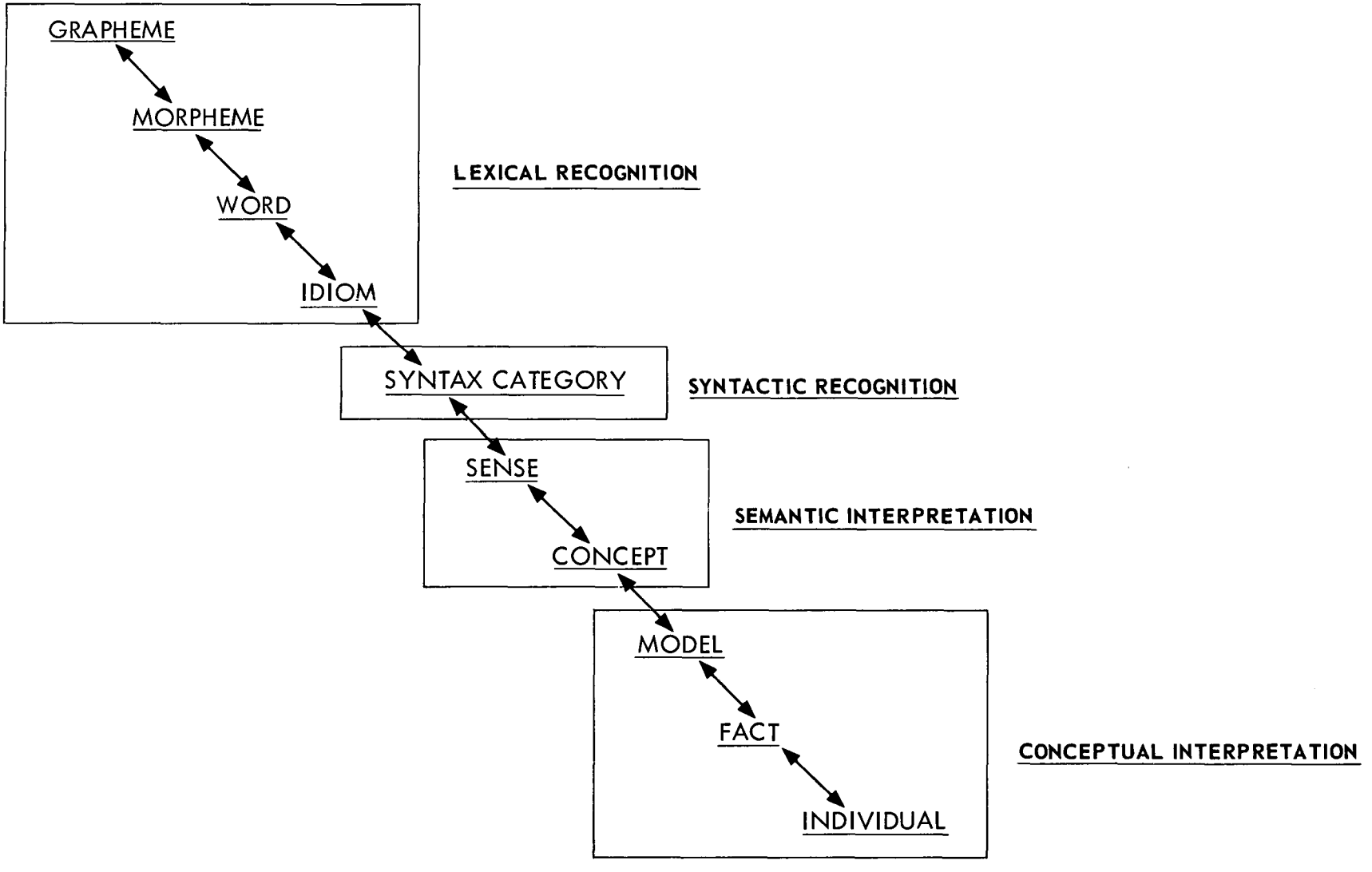


Figure 2. Levels of Recognized and Interpreted Symbolic Entities in C-360

lexicon array or "bucket" can contain a maximum of 100 lexical entries. The hash-coding scheme that is used can be adjusted to map all incoming words into from one to a present maximum of 256 buckets. This allows for a dictionary approaching 25,000 entries. Lexical recognition of the words in a sentence is always accomplished by making the minimum number of accesses to buckets in the input dictionary. Signatures permanently reside on disc storage as list structure but as many as available space permits are kept in core memory at all times to ensure quickest access to the most frequently used signatures. This dictionary organization efficiently implements the many-to-many associations between lexical items and signatures and the many-to-many associations between signatures and the concepts and objects they refer to. For example, only one signature is necessary for all members of a class concept no matter how many objects in that class are specified by proper names in the dictionary.

The present dictionary consists of about 500 lexical entries, of which about 200 are function words. Open class words such as nouns, verbs, and adjectives may be added to the dictionary by inputting sentences using entries already existing in the dictionary to define their meanings. Where possible, semantic links for new entries, class inclusion, etc., will be added automatically as open class entries are defined.

A morphological (affixal analysis) procedure will be implemented as soon as temporary core storage restrictions are overcome. Affixal analysis is important in the development of natural language processors like CONVERSE for two main reasons: (1) it reduces the size of the dictionary used by the processor insofar as it eliminates the need to store signatures for any word whose sense(s) can be predicted from signatures stored for its stem and affix(es); (2) it enables the processor to interpret productive uses of affixes. For example, the affix "-like" is productive in the phrase "CONVERSE-like processor". A proper semantic reading can be built up computationally even though "CONVERSE-like" has probably never occurred before. Any mechanical processor interacting with a human language user will sooner or later encounter word forms which it can only interpret with the help of routines for analyzing productive uses of affixes. Information about productive affixes, their morphographemic variants and the meaning changes that accompany the use of each affix is being worked out in cooperation with the "Semantic foundations for question answering" project at UCLA. John Olney, principal investigator on that project, is producing this information by processing machine-readable transcripts of Webster's Seventh New Collegiate Dictionary and The New Merriam-Webster Pocket Dictionary.

Parsing--syntax analysis and semantic interpretation: In C-32 we used a top-to-bottom, left-to-right parser that produced the first parsing permitted by a series of phrase-structure rules that incorporated checks on the semantic compatibility of constituents.

The syntactic analyzer used in the C-360 system employs a bottom-to-top, right-to-left algorithm that produces all possible parsings in a single pass. Syntactic representation is provided by two basic types of rules: structure building (SB) and structure changing (SC). The latter perform operations of a transformational type while the former have in part context-sensitive phrase-structure characteristics as well as a subset of the expressive power available in transformational rules. While the SB rules are nondirected, the SC rules are called from within the SB rules. This partial directedness allows the two types of rules to operate in an interleaving fashion so that a deep structure representation is built up simultaneously with a surface structure representation.

SB rules perform two distinct types of operations. First, they build a new node in the surface structure, complete with features and blocking symbols. And second, they use the deep-structure bracketings found on the surface nodes under consideration to build a new deep labelled bracketing. Thus the output of a SB rule is a derived signature consisting of a series of properties associated with values representing the surface syntactic category, syntactic features, and the deep-structure tree built to that point. The input of the rule is one or more lexical or derived signatures.

Blocking features, which may be put on either the derived signature or the input signatures, are widely used in the surface structure parsing to abort inadmissible structures. Additional power is gained by checking contexts to the right as the parse is constructed.

SC rules operate upon the deep structure trees created by the SB rules, converting them to other trees which become the input to still other SC or SB rules.

Among the advantages of this algorithm are the following. (1) A greater share of the work load is performed by the simpler SB rules than in a parser having a set of transformational (T) rules operating on the output of a set of phrase-structure (PS) rules; i.e., any structure building which involves copying signatures currently being operated on or creating previously non-existent signatures can be formalized by an SB rule. (2) The SC rules within the cycle can be ordered, permitting the generalities concomitant with that power. (3) The SC rules which operate within a cycle need be called only after Ss or NPs, to which they are applicable, are built. (4) Deep and surface structure information are both present when the time comes to provide a semantic representation.

Selection and sequencing of the nondirected SB rules is carried out by a process similar to that used by Kay (5) in his powerful parser. Some structure building is accomplished by methods comparable to those described by Woods (12).

The illustration in figure 3 gives a rough indication of the structures resulting from various rules of the analyzer. The structures labelled SS (surface structure) and DS (deep structure) are the final representations, the other structures are intermediate steps. WH BACKING is an SC rule which moves a relative or interrogative pronoun into its underlying relation to a following verb. Thus "in which" is moved to a PP relation with the verb "locate". The SC rule WH REPLACEMENT operates on relative clause structure to replace the relative pronoun by the noun which is "head" of the relative clause. In this case "state" is moved inside the PP "in which". AUX INVERSION moves the auxiliary or modal found at the beginning of a question into underlying position with the main verb. The SC rule SUBJECT PLACEMENT permutes the subject of an S# so that it follows the predicate with which it is associated. This permits a deep structure in which the verb is the head of a cluster of actants having various case relationships to that verb.

The English structures that the syntax analyser now parses well include declarative, imperative, and yes/no and WH questions which contain equational and verbal predicates, restrictive and nonrestrictive relativization, conjunction, superlatives, left-dislocated phrases, subjoined sentences, and clefting. The following forms can be parsed but the structures built at present are not completely satisfactory: complex determiners (quantification, partitives, and genitives), negation (determination of scope), and comparatives. Pronominalization, reflexivization, proverbalization, gapping, pseudoclefting, tag-questions, extraposition, and complex embeddings of sentences will receive attention in the near future.

The deep structures become the input to a semantic component which builds an intermediate language (IL) procedure. This component performs several functions. (1) It tests syntactically well-formed structures for semantic well formedness, eliminating those structures not passing the requirements set up by the concept network. (2) It adds quasi-semantic case information to the well-formed syntactic structures. (3) It permits interpretation rules to build up semantic representations for each node in the syntactic structure, ultimately providing a semantic representation for the whole sentence.

Briefly, the function of the concept network during parsing is to provide general information about a universe of discourse and limitations, in that universe of discourse, on the permitted states of affairs. Since concepts are defined in terms of their associations with other concepts, a multilevel, conceptual inference-making apparatus is made available to allow for checks on the semantic compatibility of syntactically related constituents and to derive, where necessary, the concepts essential to the process of IL procedure construction.

Since syntax analysis and semantic interpretation are complex processes and an apparently minor change in a rule can have wide repercussions, we make extensive use of a file of sample sentence types during the on-going process of grammar testing and extension.

Representing messages and requests as IL procedures: The C-32 intermediate language design was strongly influenced by the structure of existing artifical query languages and the predicate calculus. There has certainly been some carryover of these influences in the design of the C-360 IL. However we have been influenced also by the structure of the fact file, by set theoretical considerations, and by the desire to integrate the IL into the system as an extension of the LISP programming language. Prime motivating factors have been (1) ease of IL generation from English deep structures and (2) efficiency of execution of IL operations against concept extensions in large fact files. The intermediate language can be characterized as a variable-minimizing* lambda calculus. It is a lambda calculus--a calculus for applying functions and defining new ones--as it is an extension of the LISP programming language. The variable minimizing feature has been prompted by efficiency reasons and by a desire to preserve, wherever possible, clues of semantic or inferential significance in a form close to that used in natural language discourse.

Figure 4 illustrates several translations of sentences into our IL. The first two examples show how simple declarative sentences map into elementary storage procedures. In the first case, the domain D or the "city" concept will be enlarged to include a 1-tuple standing for the quoted object "Alameda". In the second, the 2-tuple <Arcadia, 100000> will be used to update the domain and neutral case arguments of the extension of the "population" concept.

Example three illustrates a procedure that will derive the set of large western cities by means of the intersection operator (I) determine if this class is included in the class of smoggy things (IMP), and report yes or no (TF). In the fourth, we first find the land area for Los Angeles (N = Wh) and then use it to find those cities with a land area larger than (GR) that of Los Angeles. In this case the return to TF will be a set of zero (no) or more (yes) N-tuples since no Wh-restriction has been specified at the top level.

Example five illustrates the use of a standard LISP function (quotient) in conjunction with "return" expressions. LISP functions are used to expand the range of IL expressions to give the user a direct means of requesting computational operations over derived data and to implement the process of user extension of the English subset at the lexical, semantic, and syntactic level.

*Quine (10) has shown that variables are theoretically dispensible. There seem to be sound practical reasons, however, for not eliminating them entirely.

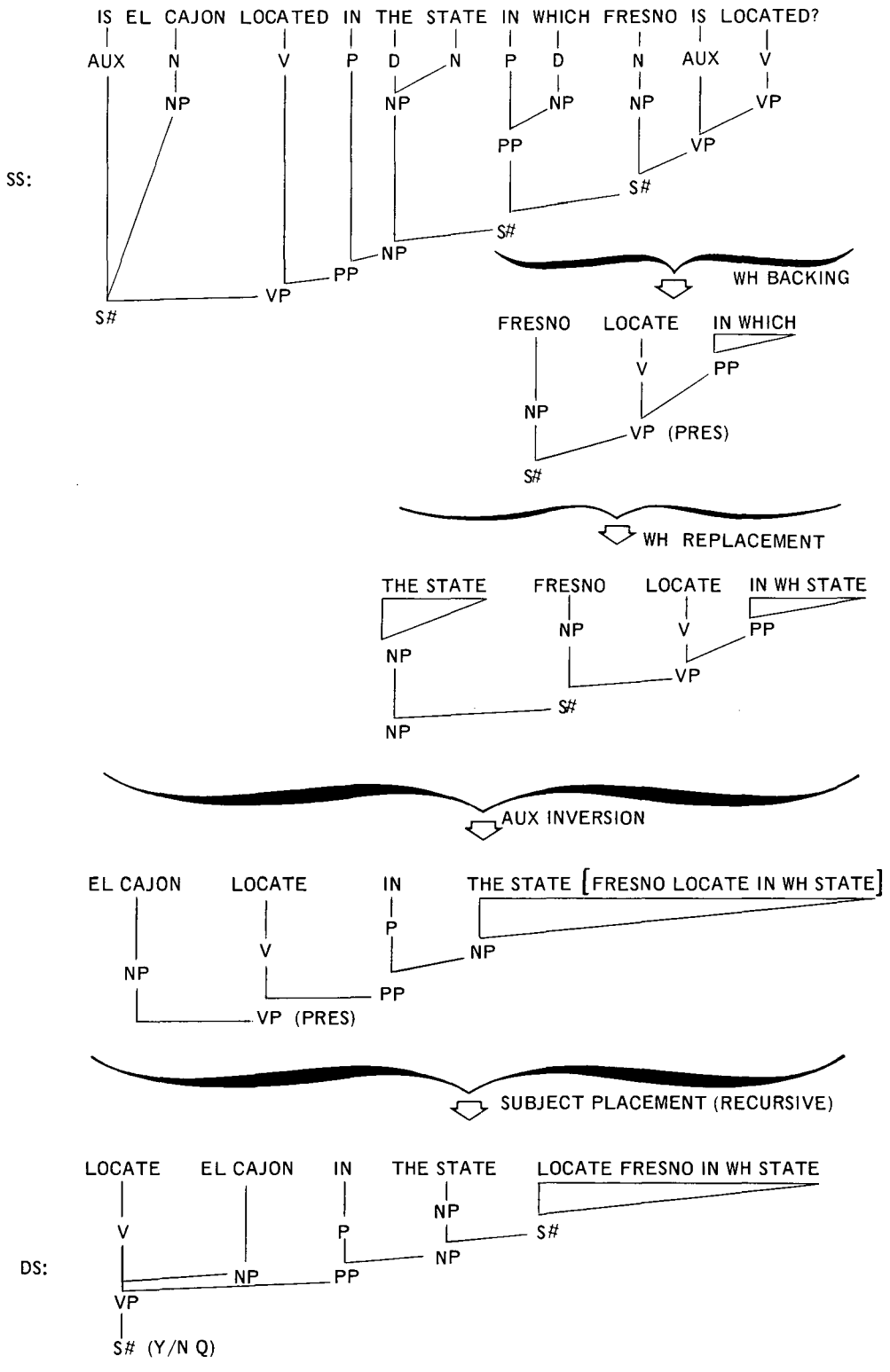


Figure 3. A Sample Surface Structure (SS) and Deep Structure (DS) Syntactic Analysis

1. Alameda is a city.
   (store (city (D = ("Alameda))))

2. The population of the city of Arcadia is 100,000.
   (store (pop (D = ("Arcadia)) (N = 1000000)))

3. Are all large western cities smoggy?
   (return (TF (IMP (I large (I western city)) smoggy)))

4. Are there any cities larger in land area than Los Angeles?
   (return (TF (land area (D = city) (N GR (land area (D = ("Los Angeles))
   (N = wh))))))

5. What is the population of Los Angeles divided by its land area?
   (quotient (return (pop (D = ("Los Angeles)) (N = wh))) (return (land area
   (D = ("Los Angeles)) (N = wh))))

6. What cities in California except Los Angeles and San Francisco have
   populations over 100,000?
   (return (I (RC (locate (D = wh city) (loc in ("California))) ("Los Angeles,
   San Francisco)) (pop (D = wh) (N GR 100000))))

7. What cities are located in states having populations less than that of
   the city of New York?
   (return (locate (D = wh city) (loc in (I state (pop (D = wh) (N LS (pop
   (D = ("New York)) (N = wh)))))

Figure 4. Sample English Sentences and Translations into the Intermediate Language

In the sixth example, semantic interpretation has supplied a missing relation (locate) and an RC (relative complement) operation is called for to compute the proper subset of cities of interest. The final example illustrates the procedure produced for a more complicated request. Here nested restrictions on the co-domain of the relation are required.

IV. Data management

The fact file--actual states of affairs in possible worlds: The C-32 data management system was a small experimental system, realized by a few LISP functions that built, maintained, and searched small files of facts in core memory. Since a principal aim in C-360 was not only to expand the scope of our English subset but also to permit experimentation with relatively large, richly structured data bases, we have constructed a data management system that is capable of accessing large collections of facts stored on disc memory.

The design of the C-360 data management system file structure has been guided by our major objective: natural communication with a formally specified universe of discourse. This led us to the work by logicians on methods for interpreting formal languages. In particular the logicians' notion of a model as an interpretation of a formal symbolic language has proved helpful in designing our data structure. A model or data base or state of affairs may be informally described as an ordered pair where the first member of the pair specifies a class of individuals comprising the universe of discourse and the second member is a function that maps symbols standing for predicates or concepts into sets of sequences of individuals. A model is, thus, a natural means for realizing the extensions of concepts.

A model may be explicitly realized, as shown in figure 5, as a collection of sets of ordered N-tuples. Sets of this kind are represented in C-360 as LISP arrays that are stored on a disc pack. Sets for four types of concepts are shown. The "city" and "state" sets represent class concepts--sets of 1-tuples that partition the universe of discourse. The "smoggy" set consists of 1-tuples representing a property rather than a class concept. Similarly "population," "located," and "departs" are represented by sets of N-tuples where N > 2. "Population" is an example of a functional concept as it represents a many-to-one mapping. The other examples represent relational (many-to-many) mappings.

For efficiency purposes all N-tuples are stored in a numerical array format. Each element in an N-tuple representing, for example, a physical object, an instant of time, or a floating point number is coded as a 32-bit computer word. This allows us not only to save storage space but to increase the efficiency of data base searches by ordering the N-tuples in an array according to some numerical function over the elements within N-tuples. At the present time all N-tuples in sets are sorted according to the entries in the domain. Whenever possible this order is used during retrieval to implement fast set operations by means of binary search techniques. All searching and maintenance functions have been written in LISP assembly language to further increase the efficiency of data management operations.

The data management system has been in use for some time and even though core space for carrying our operations is presently quite limited most retrievals have required times measured in seconds rather than minutes. The data base in current use comprises a file of over 10,000 facts about 120 cities in the United States. We expect to treat data bases in the future that are at least one order of magnitude larger. We believe that our data base structure is potentially applicable not only to many conventional data management areas but also to other areas such as document retrieval and especially to those requiring access to complex relational data. A major difference between our system and systems such as the Relational Data File (8), Leap (3) and Tramp (1) is the use of N-ary relations rather than binary relations as the basic elements of storage.

The concept network--allowed states of affairs in possible worlds: Several elementary concepts and some of the associations that may exist between them are illustrated in figure 6. Several instances of class, function, relation, and property concepts are shown. In general, complex hierarchies of these kinds of concepts are possible by making use of the subset and superset associative links (or "colored edges") within the network. Another important class of associative links are those called poss-property, poss-function, and poss-relation. These links specify which properties may apply to instances of linked concepts (e.g., cities may be smoggy); which functions may range over which kinds of objects (e.g., "population" applies to political subdivisions); and which relations may obtain among which concepts (e.g., an airplane may depart for a place at an instant in time).

Since relations are of critical importance in language processing they receive special attention. Each relation is linked by means of case relations to permitted conceptual arguments for that relation.

This type of concept network allows one to specify higher-order relations easily, e.g., properties of properties, properties of relations, etc. One type of associative link not illustrated in the figure is the def-property (defining property) link. For example, one possible def-property for the notion of "city" could be specified as: $(Pop(X) \geq 25{,}000)$.

Our concept network resembles that of Quillian (9) at the very general level of allowing every concept to point to other concepts, which point in turn to other concepts, etc. However our con-

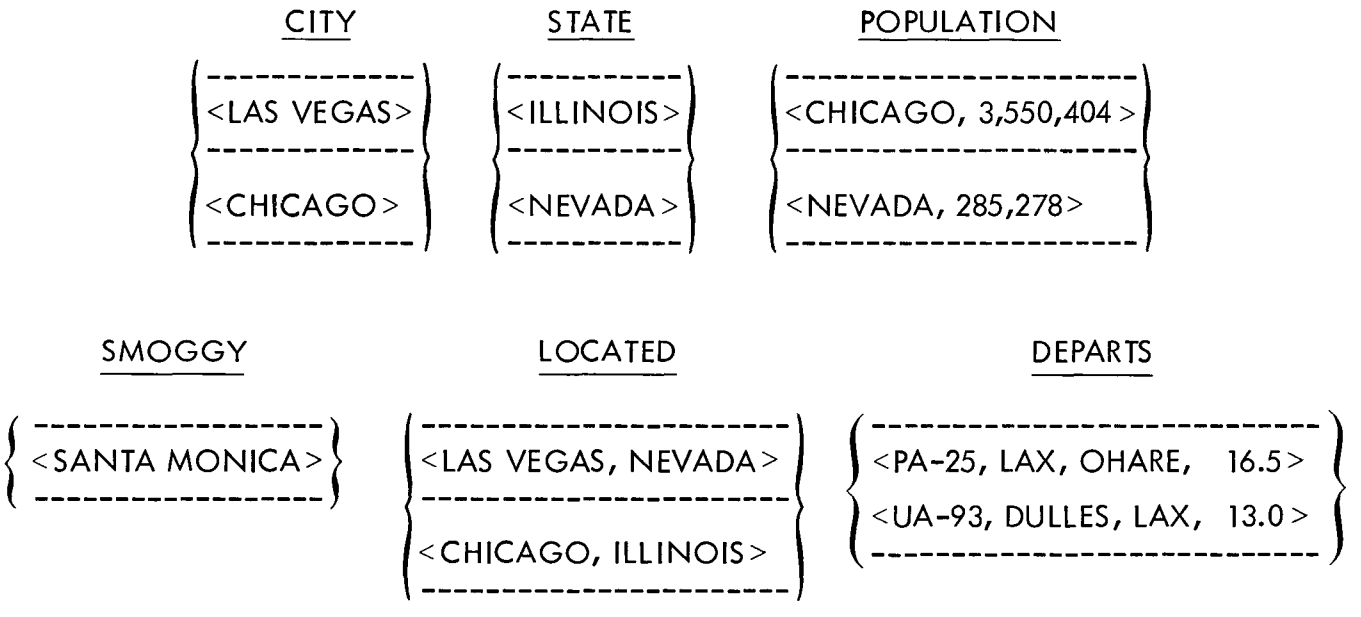


Figure 5. Elementary Specific Facts in a Fact File Model of a State of Affairs

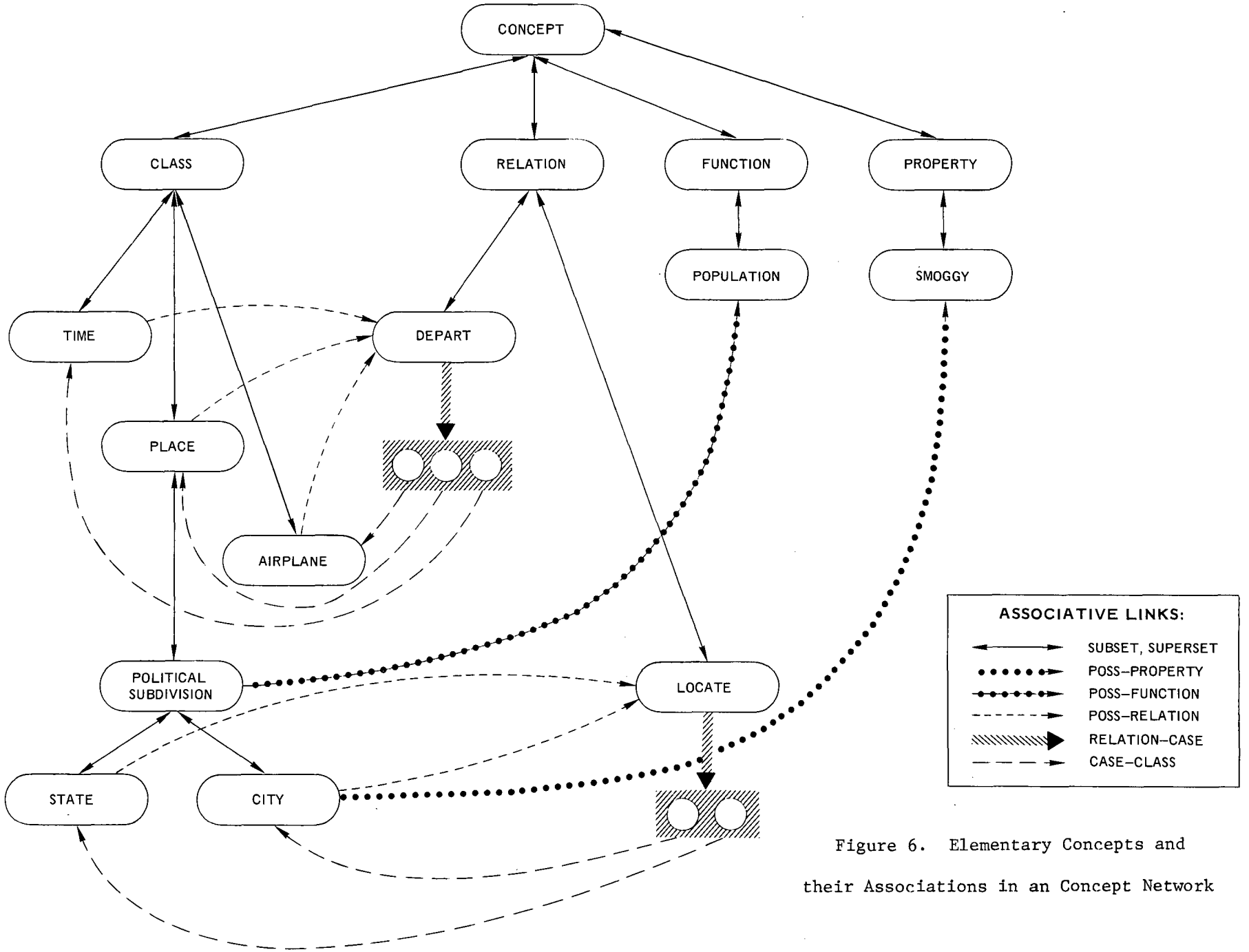


Figure 6. Elementary Concepts and their Associations in an Concept Network

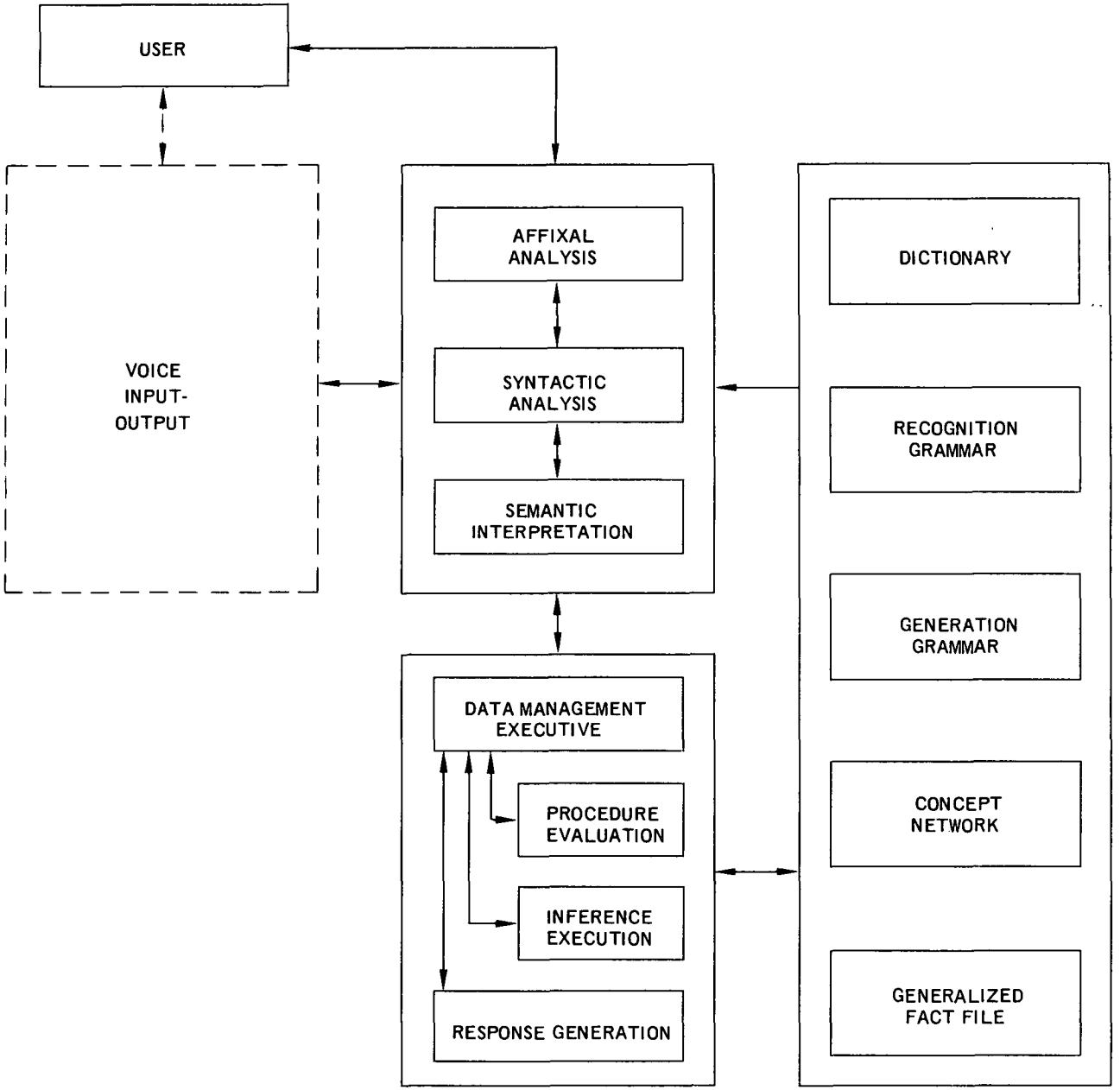


Figure 7. Advanced CONVERSE system components

ceptual structure and the kinds of associative links allowed differ from Quillian's as our network has been designed to support logical inference and semantic interpretation as an integral part of a larger language processor. At present our concept network is realized in the file of signatures in association with other semantic and syntactic information. In this form it is quite useful and accessible for purposes of semantic interpretation during sentence-to-procedure translation.

Modalities, embedded propositions, and general facts: The use of N-tuples avoids the problem of forcing the embedding or nesting of information in an artificial way as happens often in systems based only on binary relations. However embedding does occur in natural language and hence it must be dealt with. Take for example the following sentence: "Jim believed that Joe scheduled Jack to fly to Cuba on Monday." If we store the low-level proposition "Jack - fly Cuba - on Monday" we must be careful not to allow it to be retrieved as a fact. To handle this and related situations we are currently working on methods to store, identify, and retrieve modal, embedded, and general information within our data management system. Briefly this entails the association of labels, back pointers, and modal data with N-tuples. A number of important uses for modal information have been identified. For example the credibility or reliability of the sources for particular pieces of information stored within the fact file could be indicated in this way.

V. Summary and plans

At this point in the development of the CONVERSE C-360 system we are rapidly expanding the grammar and as a wider range of deep structures become available they influence our efforts in semantic interpretation, extension of the intermediate language, and enrichment of the concept network and fact file. While we are well aware of the enormity of the tasks that lie ahead in natural-language processing and keenly frustrated by the limitations of current equipment and, often, our own understanding of the problems, we cannot help but feel a sense of cautious, qualified optimism based on the fact that at least some of the major stumbling blocks are gradually being overcome.

To place the above comments on the C-360 system in the broader context of our research, we have included figure 7 in this paper. This figure illustrates the components of a future version of CONVERSE. All of the illustrated components are being actively worked on at the present time. The effort on affixal analysis, mentioned previously, will soon lead to the coding of an affix stripping routine.

The existing system already has a significant capability for effecting inference via the concept network, operations that make certain information explicit in deep structures, and IL requests that derive implicit facts from explicit facts in the fact file (see example seven in figure 4). However, it is clear that much more can be done in carrying out inferences over both specific and general facts. Larry Travis, of the University of Wisconsin, has been working jointly with us for some time to formulate and design an approach to inference making that will be capable of achieving useful deductions over large files of facts in a not unreasonable amount of computing time.

Another important area concerned with further extension of the man-machine communication interface is the work being done by J. A. Barnett and his associates at SDC on voice input and output. We are striving for a mutually beneficial relationship in which CONVERSE can benefit from speech-derived information (prosodic features, etc.) and voice I/O can use the syntactic and semantic processing capabilities of the CONVERSE system.

BIBLIOGRAPHY

(1) ASH, W.; SIBLEY, E. An interpretive associative processor with deductive capabilities. In: 23rd National ACM Conference, Proceedings. Brandon Systems Press, 1968, p. 143-156.

(2) CARNAP, R. Meaning and Necessity, second edition. University of Chicago Press, Chicago, Ill., 1956.

(3) FELDMAN, J.; ROVNER, P. An Algol-Based Associate Processor. ACM Communications 12:8 (August 1969) p. 439-449.

(4) KATZ, J. The Philosophy of Language. Harper and Row, New York, 1966.

(5) KAY, M. Experiments with a powerful parser. In: Second International Conference on Computational Linguistics, Grenoble, August 1967. Paper No. 10.

(6) KELLOGG, C. A natural language compiler for online data management. In: 1968 Fall Joint Computer Conference Proceedings. Vol 33, part I, p. 473-492.

(7) KELLOGG, C. Data management in ordinary English: Examples. In: Computer Studies in the Humanities and Verbal Behavior. Mouton, The Hague, V.I/NR.41 Dec. 1968, p. 159-182.

(8) LEVIEN, R.; MARON, M. Relational data file: A tool for mechanized inference execution and data retrieval. RAND Corporation, Santa Monica, California, 1965. (RM-4793-PR)

(9) QUILLIAN, M. Semantic memory. In: Minsky, M., ed. Semantic Information Processing. MIT Press, Cambridge, Mass., 1968.

(10) QUINE, W. Variables explained away. Proceedings of the American Philosophical Society, 104:3 (1960), p. 343-347.

(11) STOCKWELL, R.; SCHACTER, P.; PARTEE, B. Integration of transformational theories of syntax. USAF Electronics Systems Division, Bedford, Mass., Oct. 1968. Vols. I, II.

(12) WOODS, W. Transition network grammars for natural language analysis. ACM Communications 13:101 (October 1970) p. 591-606.

CUE: A PREPROCESSOR SYSTEM FOR RESTRICTED, NATURAL ENGLISH

David B. Loveman*, John A. Moyne, and Robert G. Tobey**
International Business Machines, Cambridge, Massachusetts

ABSTRACT

CUE, an input interface system which permits the computer to utilize natural but restricted English as input, is presented. In addition, an experimental model for CUE, Proto-RELADES, which can "understand" and execute English sentences about the content of the library at IBM's Boston Programming Center is described. These sentences can be query, command, or conditional sentences. The linguistic component of the system is based on a transformational grammar of English that performs a full syntactic and semantic analysis of each input sentence and translates it into relevant computer operations. The capabilities and limitations of this system are described.

KEYWORDS AND PHRASES

natural language, syntax, semantics, grammar, parsing, computational linguistics, CUE, RELADES, English

I. Introduction

The CUE system makes it possible to invoke or query a computer using natural English. CUE is an input interface between a human seated at a typewriter terminal and an arbitrary computer system. It provides immediate and natural communication by letting the Computer Utilize English input.

The phrase "input interface" describes the role CUE plays in an information system. CUE translates English input into statements in the higher level language of the information system. That system is then invoked to perform the required information processing. One possible use of CUE which highlights this aspect of the system is as an English input backup for the professional user of the information system. If he forgets the "professional language" syntax for invoking a system function, the user may lapse into an English sentence, which is then preprocessed by CUE to get the professional language equivalent. Figure 1 illustrates this role and emphasizes that CUE constructs input files for the information system. Actual execution of the host system may not be invoked immediately. (If it is more practical for a large number of queries to be gathered and then processed together, this can be done.) Output may or may not be via the input terminal. (It could be mailed to the user!)

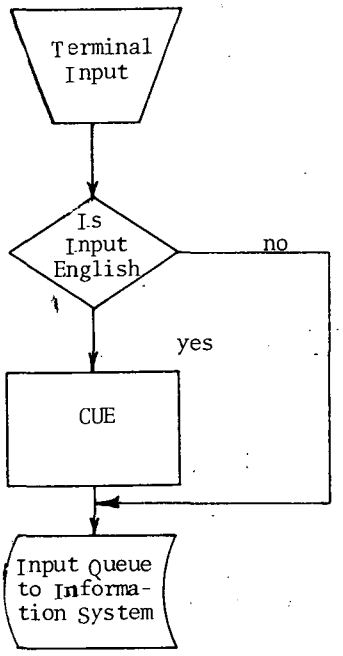


Figure 1. The CUE Input Interface

Since different host systems will require different implementations of CUE (the semantic components will be different), CUE is a generic name for many possible, specific implementations. This point is further emphasized by the simplified CUE flow diagram of Figure 2. The four modules are all table driven. Hence, any one can be readily altered to Customize the Use of English. The dictionary and semantic modules will be altered to accomplish most customization. Changes in the phrase structure and transformation modules will reflect the results of our continuing research in linguistics and in recognition grammar design and implementation.

*current address: School of Systems and Logistics, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio
**current address: Applied Mathematics Division Argonne National Laboratory, Argonne, Illinois

The goal of CUE is to make possible the use of restricted, natural English as input to a computer. Since English language processors (or processors for any natural language) have not previously been available, the way they should be used and the applications for which they are valuable need to be explored.

Salient points which cannot be overemphasized and which stress the desirability of CUE input to information systems include:

1. English input will greatly increase the use of such systems (the novice user can establish effective communication with the system immediately) and decrease wait time while greatly eliminating the need for the traditional "middleman"-the librarian or information retrieval expert.

2. Natural English input isolates system users from system changes and reduces the need for systems liaison activity. When the mode of communication is via English, most users do not have to be kept up to date with vast documentation describing the latest system revision.

3. English language preprocessors can greatly aid in the formation of an information network. Using CUE-type processors, a single request may generate special query instructions for three or four different information systems thus gaining the advantage of an information network while giving the user the sense of dealing with a single information source.

Historically, many projects have made contributions to the processing of English by computers. Although CUE is a natural outgrowth of this work (see References 1, 5, 7, 11, 17, and 19) the following linguistic and programming points are central in distinguishing CUE from other projects:

1. The grammars developed (as well as those under development) for CUE are based on transformational theory in linguistics but they are recognition grammars specifically designed for analysis of English input sentences. We take the position that a recognition transformational grammar is not simply a reversal of a generative grammar, but must be designed and implemented independently.

2. The grammars are as general and as powerful as the present and evolving state of the art permits, but to provide practical systems the semantic interpretation is narrowed to various individual fields of discourse. Thus, at various points in the syntactic and semantic analysis a recourse is made to the semantic primitives to resolve ambiguities and obtain unique interpretations. Each field of discourse or application has its own set of semantic operations in the form of computer programs or macros which constitute its universe of semantic primitives.

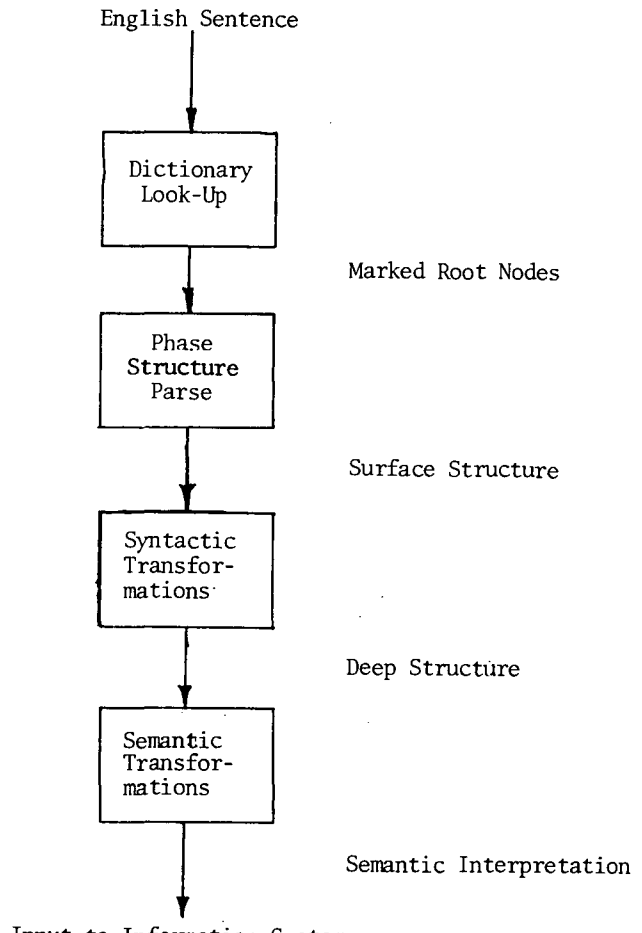


Figure 2. Simplified CUE Flow Chart

3. As far as we know none of the systems cited in the references embody a "complete" system in the sense of having (a) a general grammar which can carry out analysis and interpretation of sentences and (b) a semantic component which invokes the actions specified by input sentences.

It should be pointed out that no complete grammar for English, or any other natural language, exists which can handle the language in a general and completely unrestricted way. It is not intended to develop for CUE a general and complete recognition grammar. This would be impractical and unrealistic. The key to CUE feasibility is a project design based on developing customized grammars for limited topics of discourse. This makes it possible to resolve

most ambiguities by restricting the domain of discourse to that defined by the host information system.

It must be emphasized that CUE's ability to understand English is not predicated on simplifying assumptions as to English construction, keywords, or jury-rigging. Rather, it is based on a basic grammar and computer implementation techniques which are the object of continuing research in computational linguistics.

In the remainder of this paper CUE design is presented in greater detail, and a working model of CUE, Proto-RELADES, is described.

II. General Description of CUE

CUE is designed in two conceptually distinct modules, the syntactic and semantic analyzers. The first module, the syntactic analyzer, accepts a sentence in English, looks up the words in a dictionary, applies a phrase structure grammar to obtain a surface parse, and then applies a powerful set of transformations to obtain the deep structure. This module is general in the sense that any sentence may be given as input and, within the limits of the grammar, the correct deep structure will result.

The second module accepts as input the deep structure from the syntactic analyzer. It then applies a set of semantic transformations to determine what the sentence means in terms of the context defined by the information system. The information system then performs the indicated actions. The basic techniques used in this module are highly dependent on the information system, but they are of general utility.

CUE was designed so that the syntactic portions of the system would be as general as possible. Thus, it can be given sentences in diverse fields of discourse. For example,

> Compute x=(a+b) (a-b).
> What types of airplanes are available at Logan Airport?
> If the goods have been shipped, pay the bill.
> What treatments has Mrs. Smith received for her skin problem?
> What is the speed of rocket ABC at moment X?

If the relevant words are in the dictionary, CUE can syntactically analyze the above sentences. The system will not be able to extract the meaning of the sentences unless the appropriate semantic transformations are present, and it will not be able to perform the indicated action (compute, lookup, pay, calculate) unless the appropriate information system is provided.

A flow chart of CUE is given in Figure 3. The dictionary entries, phrase structure rules, linguistic transformations, and semantic transformations are each written in a higher level language appropriate for expressing such linguistic constructions. For each of these there is a processor which will accept input in the appropriate higher level language and translate it into an internal form suitable for efficient execution. This resulting internal form module is saved on disk. Since this translation is done once, before using CUE, considerable efficiency is gained; the modules do not have to be interpreted for each sentence. Also, the work of system extension and maintenance is made considerably easier by the use of the higher level languages. Since CUE is designed as a preprocessor to translate English into the specialized language of an information system, provision is made for a user to utilize this "professional" language directly, with no preprocessing. When a sentence is input to CUE, if it is in "professional mode" it is passed directly to the information system. If the sentence is not in "professional mode" it is linguistically analyzed.

The syntactic analyzer consists of three phases, dictionary lookup, parse, and transformations. Each word of the sentence is looked up in a dictionary where its syntactic part of speech and semantic meaning are determined. If a word is not in the dictionary, assumptions may be made as to its definition or, optionally, the user may be asked the meaning of the word ((1) on flow chart). The syntactic structure of the sentence relative to a phrase-structure grammar of English is then determined. If the input is found not to be a grammatical sentence, an error message is generated and the user can be asked to rephrase his request ((2) on flow chart). This structure is then transformed into a deep-structure by a set of syntactic transformations.

Following syntactic analysis is semantic analysis. The syntactic structure of the sentence and meanings of the words are known; what is to be determined is the meaning of the sentence in terms of the information system. This is done by applying semantic transformations to the deep-structure. In this module sentences which are correct English but meaningless in terms of the information system are detected and appropriate messages given ((3) on flow chart). The final action of the semantic analyzer is to invoke the information system itself.

III. Description of Proto-RELADES

A complete CUE prototype called Proto-RELADES (REstricted LAnguage DEfining System) is now operational at the Federal Systems Center's Boston Programming Center. This prototype allows a user to query, in English, a data base consisting of a card catalog for the Boston Programming Center library. This data base was selected for two reasons:

1. A complete card catalog prepared in the format required for the KWIC system was already available on tape, and thus, the prototype

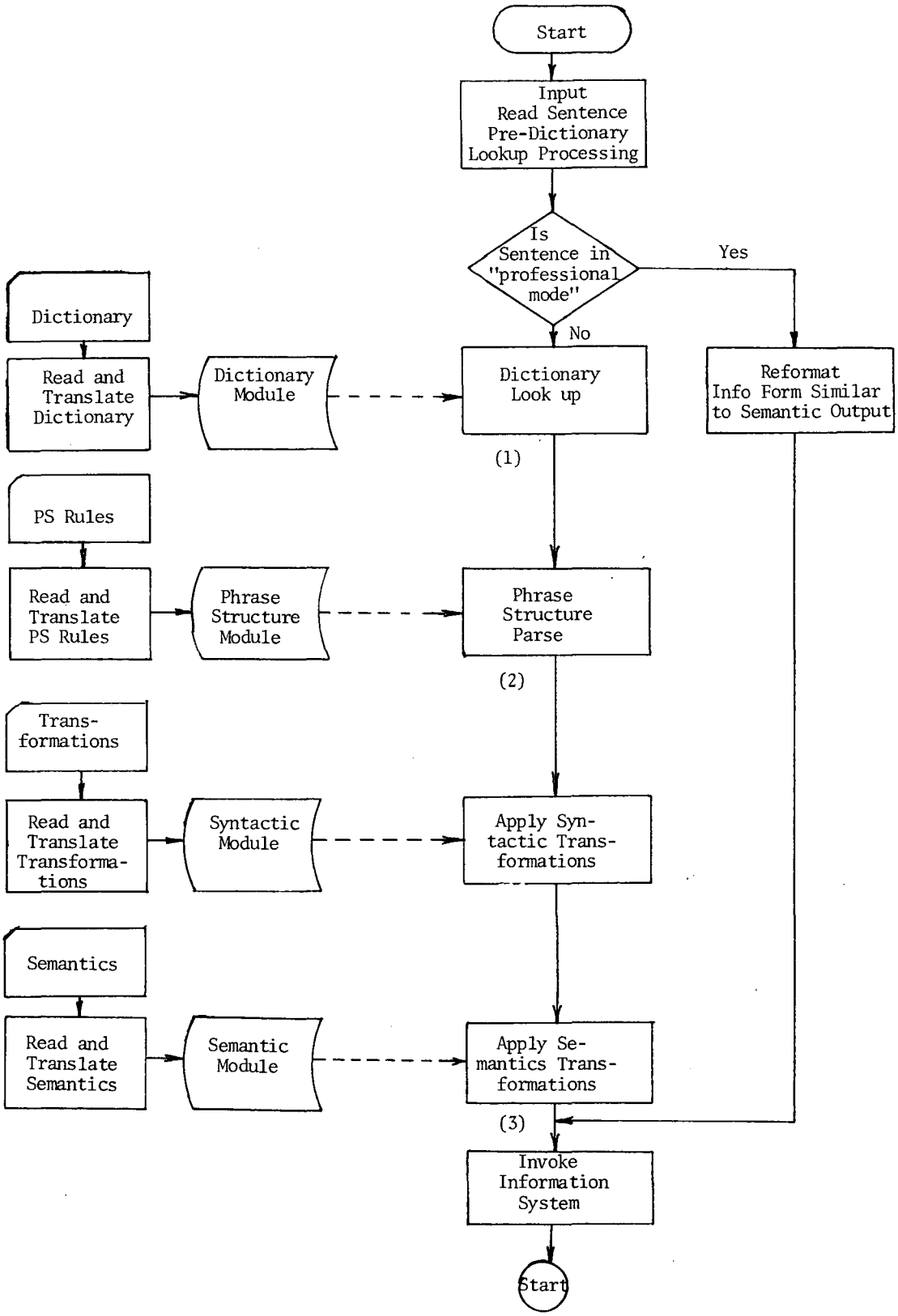


Figure 3. Overall Flow Chart for CUE

could be implemented without preparing a data base;

2. Proto-RELADES could be used for practical library operations by the members of the Boston Programming Center.

The card catalog contains entries for 3600 documents in the library. Since each entry is recorded on four to six IBM cards, the size of the current data base is over 18,000 cards. The keyword index generated contains over 34,000 keywords. This size is adequate for testing the model; however, the data base is not complex enough to demonstrate the power of the language processor.

Each entry in the card catalog of the library contains the following information:

a. Author
b. Title
c. Publisher
d. Place of publication
e. Number of pages
f. Date of publication
g. Document number
h. Volume number
i. Miscellaneous information (hard copy, revised, etc.)
j. Owner

In addition to the card catalog, Proto-RELADES has three indexing files available for the catalog to make the particular search operations more efficient. These files contain keywords, authors, and titles, along with the necessary indexing information to find the corresponding bibliography entry. The user can ask questions or give commands pertaining to the above information in the data base.

At present, the speed of the system is just under one minute for complete processing and printing of a response to an average English sentence. See Appendix A for a sample response. However, the implementation of Proto-RELADES was constrained by limited time and manpower. The primary language chosen for implementation was PL/I. First, a set of PL/I list processing subroutines, called ELF (Elementary List Functions), was designed and implemented (1). ELF is coded in PL/I, and CALLs to ELF can be freely intermixed with ordinary statements. It is a powerful and convenient list processing language with special additional functions for linguistic work. All of the linguistic programs for Proto-RELADES were written using ELF. The transformational and semantic programs were written in an interpretive language which was implemented in PL/I with the use of ELF. Working with such a hierarchy of languages embedded in each other has naturally hampered the efficiency and speed of the system. However, the speed, for a prototype model, is satisfactory, and the flexibility gained by the interpretive nature of the transformations and the grammar make it relatively easy to change either the dictionary or the phrase structure, transformational or semantic rules.

In this section we present a brief technical description of the Proto-RELADES system. To make the explanation clearer, Table 1 shows how a typical sentence would be analyzed by a simple linguistic base. Table 2 gives this linguistic base which contains just a few words in its dictionary, a small number of grammar rules and a very limited semantics. Throughout this description, we shall refer to Tables 1 and 2 as examples.

Proto-RELADES is easily modifiable in the sense that all the grammar and semantic information is entered into the system as data when the system is initialized. At initialization, considerable processing takes place to allow later execution to proceed as rapidly as possible. Several programs perform the initialization. These programs not only read in the dictionary and phrase structure rules and build cross reference indexes for them, but also read in the bibliography and create the keyword, title, and author files which index the bibliography. The transformations and semantic rules, which are written in an interpretive language, are also read in and translated from a form which is easy to write into a form which is easily interpreted by the computer. Thus, when the system is run to analyze sentences, the only initialization which must be done is the reading into core memory of the dictionary, phrase structure rules, indexes for the keyword, title and author files, and the translated form of the transformations and semantics. This reading takes approximately 25 seconds when the system is initialized. Once initialized, the system processes sentences as described below.

The input routine reads sentences either from cards or from the online console, and passes the input sentence to the dictionary lookup routine in the form of a string of characters (Table 1, Item A).

The dictionary lookup routine identifies the relevant items in the sentence and looks up each item in the dictionary (Table 2, Item A). An item is either a delimiter (period, comma, blank, colon, semicolon, double quote, question mark, exclamation mark, or number sign) or any sequence of characters up to a delimiter.

If the item is a blank, it is ignored. If the item is a double quote, all text up to the next double quote is treated as a single item and is not looked up in the dictionary. All other items are looked up in the dictionary. In the present system the dictionary is kept in core memory. This is possible only because the size of the dictionary (400 words) is sufficiently small. Another version of the dictionary is kept on a direct access disk file, and lookup is directly from the disk. An index for the dictionary

Table 1

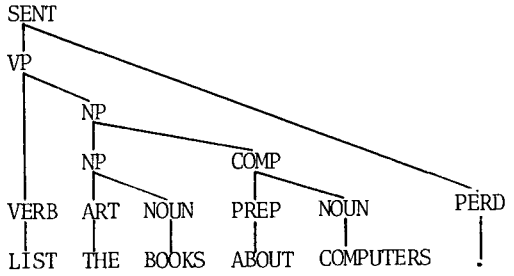
EXAMPLE USING THE SAMPLE LINGUISTIC BASE

A. Input Sentence

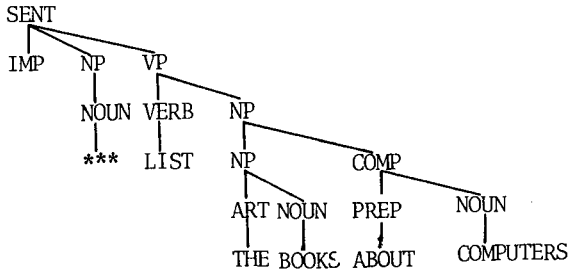
LIST THE BOOKS ABOUT COMPUTERS.

B. After Dictionary Lookup

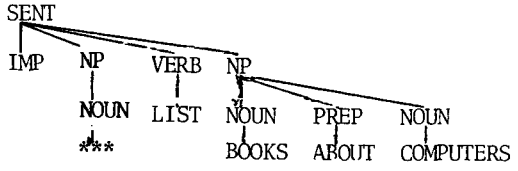
C. After Phrase Structure Parse



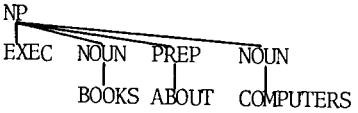
D. After Transformations



E. After Semantic Rules 1 and 2



F. After Semantic Rule 3



G. Interpretation

1) Find all items in the library with keyword COMPUTERS

2) Print all the information found

Table 2

SAMPLE LINGUISTIC BASE

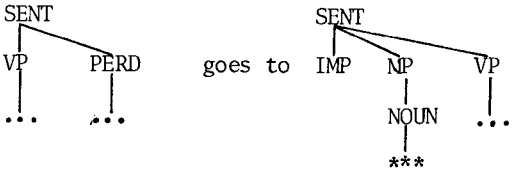
A. Dictionary

Item	Description
ABOUT	((PREP (ABOUT)))
BOOKS	((NOUN (BOOKS)))
COMPUTERS	((NOUN (COMPUTERS)))
LIST	((VERB (LIST)))
LIST	((NOUN (LIST)))
THE	((ART (THE)))
	((PERD (.)))

B. Phrase Structure Rules

Pattern		Replace By
ART	NOUN	NP
NOUN		NP
PREP	NP	COMP
NP	COMP	NP
VERB	NP	VP
VP	PERD	SENT

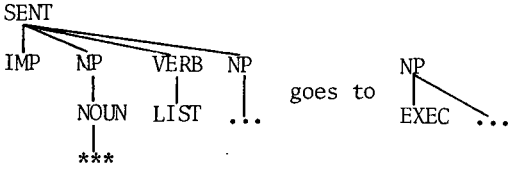
C. Transformations



D. Semantic Steps

1) Remove VP node

2) Remove extraneous structure and words under NP

3) 

E. Interpretation

EXEC means "look up and print"

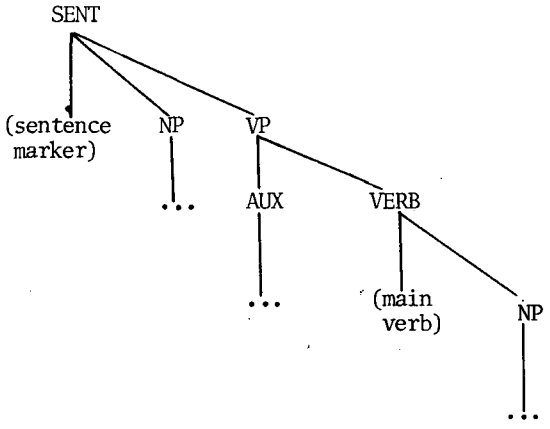
PREP
|
ABOUT

means that the following item is a keyword

is kept, giving for each letter the location of the first dictionary entry beginning with that letter. To look up an item, the index is used to find the starting point for the search. Beginning at the starting point the dictionary is searched linearly until either the desired word or one further down the collating sequence is found. If the item is in the dictionary, its meaning is determined. If the item is not in the dictionary, it is marked as NK (not known). The meaning of words not in the dictionary is determined later by the semantics. The current dictionary routine allows for a certain amount of multiple definitions of words. For example, the word "list" can be either a noun or a verb. Such ambiguities are resolved by looking at the context of the word in question.

The phrase structure parse operates on the output from dictionary lookup (Table 1, Item B) and results in the surface structure of the sentence. The phrase structure rules are partitioned and iterative (Table 2, Item B). The sentence is scanned from left to right, and rules are applied sequentially until no rule applies anywhere in the sentence. If the input was syntactically correct, there should be one top node in the tree structure resulting from the phrase structure parse and the top node should be labeled SENT (Table 1, Item C).

The transformation rules apply to the surface structure created by the phrase structure rules (Table 2, Item C). The purpose of the transformation rules is to reduce the surface sentence into a standard form deep structure. For example, consider the following three sentences: "John hit the ball," "Did John hit the ball?" and "The ball was hit by John." These three sentences all contain the same information, albeit in different sentence forms. Thus, these three sentences, although they have different surface structures, should have the same deep structure, the only difference being the sentence marker indicating what type of sentences they are. In our system the deep structure standard form is as follows:

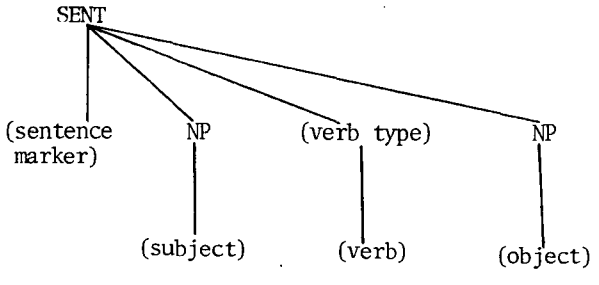


where SENT, NP, VP, AUX, etc. are the names of various linguistic items.

A transformation consists of a structural description and a series of structural changes. If a sentence is of the same form as the structural description of an applicable transformation, the sentence structure is modified as indicated by the structural changes. Transformations are scanned in order until one applies. When a transformation applies, information is gained as to the nature of the sentence, and rather than applying transformation in order from that point on, control transfers to next applicable transformation. Thus, many transformations can be skipped over because it is known that they cannot apply (Table 1, Item D).

Up to this point, the system is independent of the particular data base and nature of the data base under discussion. The linguistic analysis to this point depends only on the nature of the language-not on the nature of the area of discourse. From this point on, however, changes which depend on the nature of the data base are made to the sentence. Thus, in a sense, the rest of the system can be considered as an interpreter, which accepts inputs in the form of deep structures. Some deep structures contain meaningful instructions for the interpreter, and some do not. The ones which do not will be rejected. The ones which are rejected, for example, refer to something unknown by the data base. Were one to ask Proto-RELADES "What will the weather be tomorrow?," a correct deep structure would result but the semantics would not be able to understand what is meant by "weather", or "tomorrow."

The semantics (Table 2, Item D) simplifies the structure of the sentence based on knowledge of the data base. Each noun phrase is simplified in the sense that some structure is eliminated and all words other than those marked NOML, P or ADJ are deleted. The node AUX is deleted and the main verb and the object are pulled up to the same level as the subject noun phrase. The result of this phase of semantics is a much simplified standard form for the sentences as follows (Table 1, Item E):



The meaning of the sentence must now be determined in terms of the data base and primitive functions. Some of the operations performed by this phase of semantics are listed below.

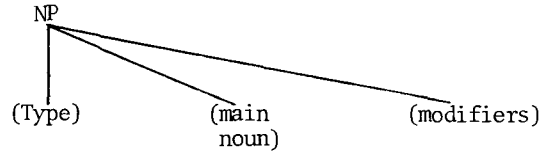
1. A check is made to be sure that the subject of the sentence is allowable; i.e., something which is known to the data base, for example, "we", "you", "library" or some other allowable word.

2. The sentence marker and main verb are analyzed to determine which primitive functions are to be invoked, and in what order.

3. A check is made to determine that the object of the sentence refers to something in the library. For example, "anything", "document", "books", "title", "author", or some other allowable word.

4. In the course of the analysis of a sentence, the grammar may generate declarative sentences (DCL) from complex query or command sentences. For example, if the input sentence is "Do you have the books which Floyd wrote?," the grammar would generate two simple sentences "Q you have books" and "DCL Floyd wrote the books." The combined structure of the input sentence and these declarative sentences are further analyzed at this point by the semantic component of the system. The information in the declarative sentence modifies some portion of the main sentence. In the above example, the DCL sentence narrows the books in the Q sentence to those written by Floyd.

The final output from the semantic steps, which is passed on to the control routine which calls the primitive functions, is also in a standard form as follows (Table 1, Item F):



Where type indicates which data base routine is to be called, main noun contains information about output formatting and modifiers contain information about parameters for the lookup search.

The interpretation routine for the primitive functions now takes over (Table 2, Item E). A search is performed to isolate a subset of the data base depending on the nature of the modifiers. It is assumed that any noun or unknown surrounded by double quotes is the title of a document. The search proceeds and identifies a subset of the documents, and returns a number which is a count of the number of documents which were found. If the type indicates a yes-no question, a message is given to the user of the form X DOCUMENTS WERE FOUND where X is the number found by the lookup. If the type is EXEC, this means the user wants some form of printout of the documents found. The nature of the printout is determined by the main noun. If the main noun indicates some particular field of the card catalog entry, such as title, place of publication, author, or date of publication, only the items requested are printed. If the main noun does not indicate this (for example, if the main noun is "books" or "documents" or some other equivalent word), the entire contents of every card for every document found is printed (Table 1, Item G). After satisfying the user's request, the system goes on to process the next request.

The above has been a sketch of the inner workings of the current Proto-RELADES System.

IV. Summary

The CUE systems for natural English input to an information system and an operational, experimental model, Proto-RELADES, are described. An actual example (see the Appendix) illustrates the linguistic mechanisims utilized. It is emphasized that two facts contribute to the practicality of the CUE approach:

1. The discourse which CUE can handle is restricted to the context defined by the host information system.

2. The CUE grammar is under constant extension and revision; it reflects the current linguistic state of the art.

BIBLIOGRAPHY

(1) BOBROW, D.G. A Question-Answering System for High School Algebra Work Problems. In: AFIPS Conference Proceedings, vol. 26; 1964 Fall Joint Computer Conference. p.591-614.

(2) CHEATHAM, T.E. On the Generation and Implementation of Reductions Analysis Programs for Context Free Grammars, Massachusetts Computer Associates, CA-6902-2911, Feb 29, 1968.

(3) CHOMSKY, N. Aspects of the Theory of Syntax, MIT Press, Cambridge, Mass. 1965.

(4) CULICOVER, P; KIMBALL, J; LEWIS, C; LOVEMAN, D; MOYNE, J. An Automated Recognition Grammar for English. IBM, Cambridge, Mass. FSC 69-5007, July 1969. 103p.

(5) DEACON PROJECT. Phrase-Structure Oriented Targeting Query Language. RM65TMP-64, TEMPO, General Electric Co., Santa Barbara, Calif. 1965
(6) KATZ, J; POSTAL, P. An Integrated Theory of Linguistic Description. MIT Press, Cambridge, Mass. 1964.
(7) KUNO, S; OETTINGER, A. Multiple-Path Syntactic Analyzer. In: Mathematical Linguistics and Automatic Translation, NSF-8. Computation Laboratory, Harvard University, Cambridge, Mass. 1963.
(8) KUNO, S. Computer Analysis of English. In: Mathematical Aspects of Computer Science. American Mathematical Society, vol. 19. 1967.
(9) LOVEMAN, D. An Introduction to ELF - List Processing in PL/I. IBM, Cambridge, Mass. FSC TM 48.68.001. Jan 22, 1968.
(10) LOVEMAN, D. Language Defining System. Unpublished.
(11) MITRE. English Preprocessor Manual. The MITRE Corp., Bedford, Mass. SR-132, Dec. 1964.
(12) MOYNE, J. Towards the Understanding of Natural Languages by Machine. IBM, Cambridge, Mass. TR BPC2. Oct 3, 1967.
(13) MOYNE, J. Proto-RELADES - A Restrictive Natural Language System. IBM, Cambridge, Mass. TR BPC3. Oct 3, 1967.
(14) MOYNE, J. Information Retrieval and Natural Language. In: Proceedings of the American Society for Information Sciences. vol. 6. 1969.
(15) MOYNE, J. A Progress Report on the Use of English in Information Retrieval. Proceedings of the International Congress of Computational Linguisties. Sweden. 1969.
(16) PETRICK, S. A Recognition Procedure for Transformational Grammars. MIT doctoral thesis, Cambridge, Mass. 1965.
(17) ROSENBAUM, P; LOCHAK, D. The IBM Core Grammar of English. In: Specification and Utilization of a Transformational Grammar, D. Lieberman (ed.). IBM, Yorktown Heights, N.Y. 1966.
(18) ROSENBAUM, P. English Grammar II. IBM Research Report RC2070, Yorktown Heights, N.Y. 1968.
(19) WEIZENBRAUM, J. ELIZA-A Computer Program for the Study of Natural Language Communication Between Man and Machine. Comm ACM vol.9, no. 1, Jan 1966. p36-45.
(20) WOODS, W. Semantics for a Question-Answering System. Report No NSF-19. The Computation Laboratory, Harvard University, Cambridge, Mass. 1967.

APPENDIX A SAMPLE ANALYSIS OF SENTENCES

THE INPUT SENTENCE IS...

HAVE ANY BOOKS ON AUTOMATA BEEN WRITTEN? #

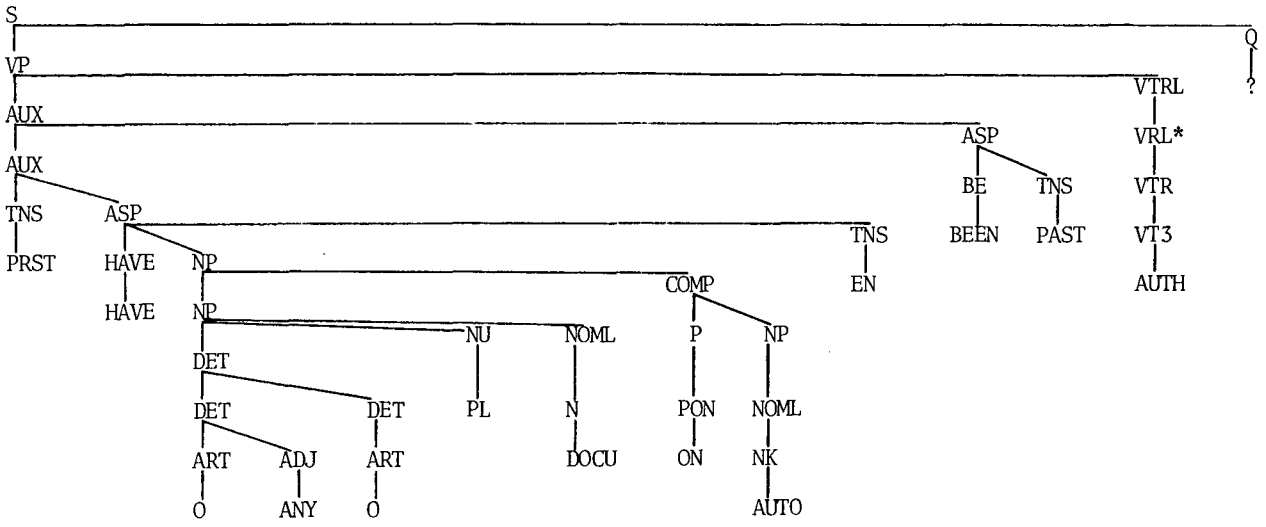
Dictionary Lookup

HAVE	((TNS (PRST)) (HAVE (HAVE)))
ANY	((ART (0)) (ADJ (ANY)))
BOOKS	((ART (0)) (NU (PL)) (N (DOCU)))
ON	((PCN (ON)))
AUTOMATA	((NK (AUTOMATA)))
BEEN	((TNS (EN)) (BE (BEEN)))
WRITTEN	((TNS (PAST)) (VT3 (AUTHOR)))
?	((Q (?)))

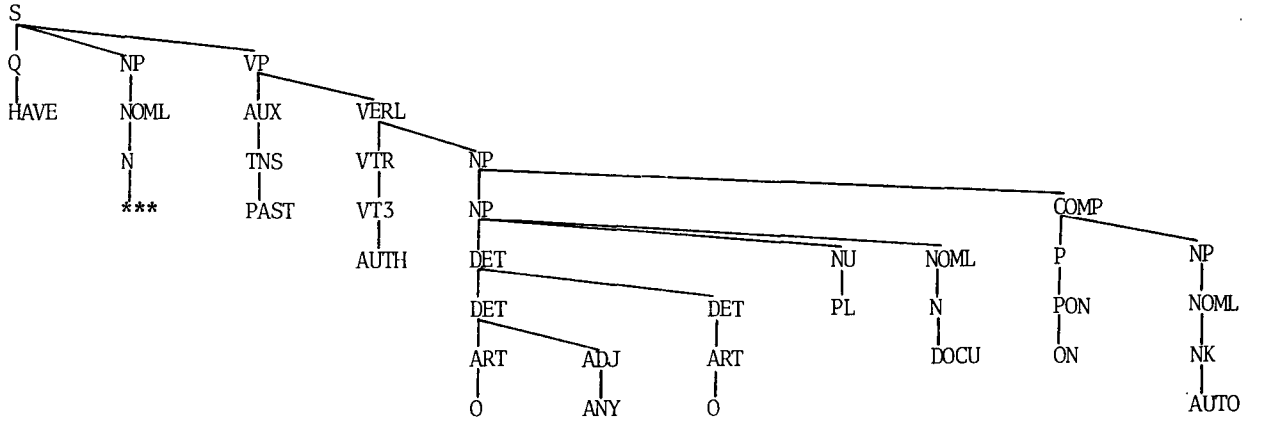
Output from the Dictionary

TNS	HAVE	ART	ADJ	ART	NU	N	PON	NK	TNS	BE	TNS	VT3	Q
PRST	HAVE	0	ANY	0	PL	DOCU	ON	AUTO	EN	BEEN	PAST	AUTH	?

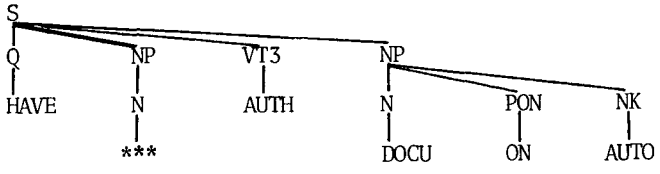
The Surface Structure: Output from the Surface Grammar



The Deep Structure: Output from the Transformational Component



Output from the Semantic Component



Answer to the Input Sentence

(In this case the input sentence was a yes-no question and the positive answer is indicated by printing the number of the documents found with the specifications indicated in the question.)

```
20          DOCUMENTS WERE FOUND
```

APPENDIX B SAMPLE SENTENCES WHICH PROTO-RELADES CAN PROCESS SUCCESSFULLY

Do we have "Modern Algebra" by Birkhoff?
Do you have anything concerning IBM?
Do you have any books which Smith wrote in the Library?
Do you have any books which Smith wrote about Computers?
Do you have any books with Keywords Computer and Conference?
Do you have any books written by Smith?
Do you have a book about Language?
Do you have any books in the Library which Smith wrote about a Computer?
Do you have what Smith wrote on Computers?
Do you have what items Smith has written?
Do you know the books which Smith wrote?
Do you know the papers which Smith has written about Computers?
Do you know the book which Smith wrote on Computers?
Do you know what items Smith has written?
Do you know what Smith wrote on Computers?
Do you know who wrote "Modern Algebra?
Do you know who is the author of "Modern Algebra"?
Do you know if Smith wrote anything on Language?
Do you know if Smith has written any books?
Do you know if the Library has any books written by Smith?
Do we subscribe to "Modern Algebra"?
Does the Library have any books written by Smith?
Does the Library have a copy of the document on Computers by Smith?
Does the Library contain any articles on Language?
What do you have concerning IBM?
What does the Library have concerning Computer Languages?
What does the Library have about Computers?
What did Gorn write?
What did Smith write on Computers?
What books about Compilers are available?
What books about the LARC are in the Library?
What documents are written by Smith?
What books has Althoff written?
What books having to do with Language do you have?
What books did Smith write on Computers?
What material do we have pertaining to Language?
What articles do we have on Computer Conference?
What documents do you have dealing with Language?
If you have any books on Computers, list them.
Have any articles on Languages been added?
Have any articles on Language been written?
Have any books on Language been written by Smith?
Has Smith written anything which we have?
Has Gorn written any documents?
Who is the author of "Automata Studies"?
Who wrote "Advances in Computers"?
Are there any books by Gorn?
Are there any documents on Language in the Library?
Is a book about Language in the Library?
Is "Modern Algebra" in the Library?
Are there any books with Keywords Computer and Conference in the Library?
Are there any books in the Library which Smith wrote?
Will you give me the title of the paper by Smith?
Could you tell me if you have any books on Computers?
Could you tell me if you have books by Smith?
Can you tell me if Althoff has written any documents?
Can you tell me if Smith wrote anything on Language?
Can you tell me what books about Management are in the Library?
Can you tell me what items Smith has written?
List all the papers by Blaauw.
List all documents about the LARC.
List our papers by Smith
List our set of papers by Smith
Look up the author of "Modern Algebra".
Give me what you have on Language please.
List all the documents on Computer and on Conference.
List all the documents on Computer and Conference.
Find all the documents on Compilers and list them.
Find all the books which Smith wrote and list them.
List all the items with Keywords Computer, Proceedings and Smith.
Give me the titles of the documents on Language which the Library has.
Give me all the books about Computers which Smith wrote.
Give me all the books about Computers which you have in the Library.
List all the documents which are about Language.

APPENDIX C THE GRAMMAR

The recognition grammar is organized according to the description given below. The areas discussed are those relevant to the linguist using the system, and they comprise what might be considered the tip of the iceburg. A detailed description of the grammar may be found in (4).

The grammar consists of the following discrete components: 1) the dictionary; 2) the Reductions Analysis (RA) rules; 3) the transformations; 4) the semantics; 5) the semantic interpretation. The sentence is conceptualized as a string of words on a tape. Each word is looked up in the dictionary and assigned to a category according to its dictionary entry. If a word is unknown to the computer the grammar attempts to determine its category from the context surrounding it during the processing of the sentence.

Following dictionary lookup the "tape" on which the sentence is written may be viewed as being divided into "boxes" with one "word" to a box. The sentence is read by the RA rules one box at a time, and each RA rule tests the input string to see if the rule is applicable. If the proper conditions for the application

of the RA rule are met by the string read in, then the rule applies, and the next rule in the ordering is tested. If the rule cannot apply, an alternate branch in the ordering is followed. By such a device each RA rule need be tested only if there is reason to suppose, by the nature of the rules which precede it, that it may apply.

The purpose of dictionary lookup, the RA rules and transformations is to produce a tree which is called the "deep structure" of the sentence. The deep structure indicates the underlying gramatical relations between the words in the input sentence. It is interesting to note that with this form of the grammar no surface structure of the sentence is produced. An application dependent set of transformations called the "semantics" modifies the deep structure into a tree which includes only those aspects of the deep structure which are revelant to the interpretation of the sentence. The semantic interpretation of the tree thus generated produces a string which is in the target language of the information retrieval system.

The reading of the sentence by the RA rules may be seen as the movement of a pointer from left to right along the boxes of the input string. Each rule applies in the immediate neighborhood of the pointer, and so it is possible for rules to apply at many different points of the sentence, whenever the context surrounding the pointer is satisfied. As a consequence , the processing of the sentence is cyclic in nature, since entire sequences of rules may apply several times if the necessary contexts are repeated several times in the sentence.

In certain contexts transformations are applicable. All or part of the context for a transformation is tested in the RA rule, and all or part of the context may be tested in the transformation itself. If the context mentioned in the RA rule is satisfied, then the transformation is "called". If the context specified in the transformation is satisfied, then the transformation applies. If the context mentioned in the transformation is not satisfied, then the processing of the sentence is returned to the RA rule from which the transformation was called, or to another transformation, if this is desired. If the transformation applies, then processing may be returned to the RA rule which called the transformation, or it may be transferred to another transformation.

An RA rule is written according to the following format:

Rn: A B C →D ¢..X..¢ @..Y..@ *** /Rm#
This may be described in the following way:

The rule has the label "Rn". If the sequence A B C is immediately to the left of the pointer in the sentence, and if the condition "..X.." (between ¢ signs) is satisfied, then perform the action "..Y.." (between @-signs) and rewrite A B C as D; move the pointer over as many boxes as there are asterisks (*) mentioned in the rule, and go to the rule labeled "Rm".

A variation of this is as follows:

Rx: A . B C ┃ D.E →F ¢..X..¢@..Y..@ *** /Ry #

The label of this rule is "Rx". The function of the rule is to check to see that A B C are to the left of the pointer in the sentence, and that D E are to the right. If this is the case then if the condition ¢..X..¢ is satisfied then the action @..Y..@ is performed and the sequence between the dots (namely B C D) is rewritten by the symbol to the right of the arrow (F). The pointer is automatically moved immediately to the right of the righthand dot, and is then moved from there as many boxes to the right as there are asterisks. Finally we go to rule Ry.

If the GO TO instruction in the rule (e.g., /Rm and /Ry above) does not appear or if the rule "failed" then the next rule in the linear ordering of the rules applies. If there is no symbol to the right of the arrow then the function of the rule is merely to check the context, to perform the action if there is one, to move the pointer if this is desired, and to go to the rule mentioned after the slash (/).

Although all RA rules are fundamentally identical, we may conceptually distinguish two types. Those rules with a label of the form "Rn", where n is an integer, are not empty of linguistic content. In other words, for most of these rules there is some real linguistic motivation. An example would be a rule such as R12: PREP NPP→ PP. Those rules which appear at the beginning of the list with a label of the form "Dn." where n is an integer, are called "dispatch rules" or "traffic rules". Their main function is to direct the processing of the sentence to those rules which perform rewrites, or which call transformations. In this way we can avoid the time-consuming checking of every rule in the list to see if it applies everywhere in every sentence. The dispatch rules enable us to ignore whole blocks of rules which cannot apply because the conditions for their application are not satisfied in the particular portion of the sentence being processed.

Some of the dispatch rules also perform a function which we may call "canonicalization". An example of this would be rule D13, considered concurrently with rule R6. The first rule provides that if a determiner does not precede those elements which will eventually form a noun phrase, then a determiner is inserted in the proper position. The second rule later rewrites DET NP as NPP, whether or not the determiner appeared originally in the sentence or was inserted by a rule such as D13. D13 guarantees that every NPP will contain a determiner, so that it will not be necessary to have an extra rule to create NPP's out of NP's which are not preceded by DET.

Another situation of this type is found in the case of R7 and R10.

The remainder of this appendix is devoted to a listing of a fragment of the RA rules and to a brief description of the function of each one. The rules in this listing are in the form in which they are entered into the computer. It will be observed that very few of the rules are at all similar to the phrase structure rules commonly encountered in generative grammars. This is due to the fact that most of the rules exist only to direct the flow of the processing of the sentence, or to insert dummy nodes which will regularize the form of the deep structure and make it possible to have a limited number of possible general deep structures. A deep structure may be as large as one desires, but the same patterns will repeat themselves again and again. The number of possible deep structure patterns has been severely limited to make the semantic processing as simple and as efficient as possible.

```
STRT:                          *#

D13:    .ANY|.N--> ¢'DET'¬=S(SP).T ¢ @ INSERT ('DET', '(DET)', 0); @/R7#

        #BEFORE A NOUN INSERT A DETERMINER IF ONE IS NOT THERE
        #GO TO THE SECTION OF RULES WHICH DEAL WITH DETERMINERS

D14:    CMPN -->               /R10#

        #IF YOU HAVE A NOUN COMPLEMENT, GO TO THE RULE
        #WHICH REWRITES N CMPN AS NP.

D15:    N -->                  /R7#

        #AFTER A NOUN, GO TO THE RULE WHICH CHECKS TO SEE
        #IF WHAT FOLLOWS IS A REDUCED RELATIVE CLAUSE BEGUN
        #WITH A PARTICIPLE.

D16:    .PREP|.TAIL --> @ INSERT ('NPP', '(NPP)', 0); @/R12#

        #IF A PREP PRECEDES TAIL, THEN INSERT A DUMMY NPP AND
        #GO TO THE RULE WHICH REWRITES PREP NPP AS PP.

D17:    PREP -->               */R12#

        #AFTER A PREPOSITION, MOVE OVER ONE BOX AND GO TO THE
        #RULE WHICH MAKES PREPOSITIONAL PHRASES

R6:  DET NP --> NPP @IF I=0 THEN DO; I=1; CALL SVINVI; END;@ /R6B#
                                /R7#
        #REWRITE DET NP AND GO TO THE TRANSFORMATION WHICH
        #APPLIES INVERSION IF IT HAS NOT ALREADY APPLIED.  THEN GO TO
        #THE RULE WHICH DISPATCHES NPP.
        #IF THE RULE FAILS GO TO THE RULE WHICH INSERTS A DUMMY
        #CMPN AFTER NOUNS WHICH DO NOT HAVE A CMPN.

R7:  .N |.  TNS      --> @INSERT ('CMPN', '(CMPN)', 0);@ /R10#
     .N |.  WH       --> @INSERT ('CMPN', '(CMPN)', 0);@ /R10#
     .N |.  TAIL     --> @INSERT ('CMPN', '(CMPN)', 0);@ /R10#
     .N |.  CONJ     --> @INSERT ('CMPN', '(CMPN)', 0);@ /R10#
     .N |.  NV       --> @INSERT ('CMPN', '(CMPN)', 0);@ /R10#
    /STRT#   INSERT A CMPN AFTER AN N WHICH HAS NO COMPLEMENT

R10:   N CMPN --> NP  /R6#  SELFEXPLANATORY.  GO TO THE RULE WHICH DEALS WITH DET NP.

R12:  PREP NPP -->PP   #

R12A: NEG PP -->  PP     /R13A#

        #IF WHAT FOLLOWS THE PREPOSITION HAPPENS TO BE A NOUN PHRASE,
        #THEN THE TWO FORM A PREPOSITIONAL PHRASE.->RULE WHICH REWRITES PP'S AS CMPN AFTER A NOUN.

R13A: N  .PP |.  -->   CMPN   /R10#

        #IF A PP FOLLOWS A NOUN, THEN IT IS A CMPN.  GO TO THE RULE WHICH REWRITES N CMPN AS NP.
```

FULL TEXT DOCUMENT RETRIEVAL: HEBREW LEGAL TEXTS

(Report on the first phase of the responsa retrieval project)

Y. Choueka[1], M. Cohen[2], J. Dueck[3], A. S. Fraenkel[4]*, and M. Slae[4]

ABSTRACT

A full text retrieval system was designed for the responsa literature, which is a large corpus of Hebrew legal cases. The unique problems of the data base — mixture of Hebrew, Aramaic and vernaculars, lack of vowels and punctuation, extreme language inflection problems, homographs, existence of thousands of grammatical variants of any given keyword — dictated development of new methods. Among them we list "grammatical synthesis", which synthesizes all grammatical variants of a given keyword; "Compact KWIC", which enables the user to have a glimpse of the nature of the search before having performed it; effective citation index imbedded in full text searches; and, in general, extensive use of both positive and negative feedback within a single search run. A number of searches performed on a relatively small data base gave in each case a recall of 100%. The average precision was 34%. A KWIC of strategic portions of retrieved documents usually enables a quick disposal of non-relevant material.

KEYWORDS AND PHRASES: Full text retrieval, metrical operators, feedback, grammatical synthesis, Hebrew computational linguistics, legal cases, responsa, case law retrieval.

1. INTRODUCTION

The responsa are written answers given by foremost Jewish authorities to questions submitted to them. They are legal cases, the equivalent of court decisions. This 17-centuries old literature is a result of Jewish self-government in the countries they have been residing in. The responsa cover all aspects of life and concern themselves almost exclusively with cases taken from real life, with questions that bother the individual or the community.

The surviving remnants of this vast literature consist of thousands of volumes, written by thousands of authors, containing hundreds of millions of words. Only very partial and incomplete indexes exist. Among the modern-day indexing projects we mention the classification of selected responsa based on the code Shulchan Aruch ("Otzar Haposkim" Vols. 1-11, Jerusalem, 1968); the hierarchical index being constructed now by the Institute of Research in Jewish Law at the Hebrew University ("Index to the Responsa of Rabbi Asher ben Yechiel", M. Elon editor, Inst. Res. Jewish Law, Hebrew University, Jerusalem, 1963); and the card catalog index to the historical material of Middle-Eastern responsa being prepared in the Department of Jewish History at Bar-Ilan University. (For other references, see S. B. Freehof, "The Responsa Literature", The Jewish Publication Society of America, Philadelphia, 1959).

Since the corpus, which is largely inaccessible today, contains a great wealth of material of interest to jurists, historians, linguists (e.g., development of language), sociologists, economists and many others, we decided to design a document retrieval system for it.

Work was started in late 1967, and the first experimental phase was concluded in the middle of 1969, according to plan. The unique nature of the data base presented challenges not encountered in similar undertakings where the data base is in English. To meet these problems, we developed a number of methods believed to be new, and incorporated them into a single integrated retrieval system.

Although these methods were developed to solve our particular problems, the ideas underlying them are of sufficient generality to be applicable directly to projects with data bases in English, where they are expected to improve the recall and precision simultaneously as was the case with our project. Therefore we believe that this work is of interest also to the English-speaking community, and we consequently decided to translate, amend and enlarge our one-and-one-half year old final report in Hebrew (8). The result is this paper. See also (7,12), where some of our work has been described.

We are currently in the midst of our second phase of development of the project, during which we will increase our data base and run 100 test searches. But this report concerns itself almost

(1) Department of Mathematics, Bar-Ilan University.
(2) The Inter-Kibbutz Computer Center, Tel-Aviv.
(3) Institute of Research in Jewish Law, The Hebrew University of Jerusalem.
(4) Department of Applied Mathematics, The Weizmann Institute of Science, and Department of Mathematics, Bar-Ilan University.

* Work supported, in part, by the U.S. National Bureau of Standards. Reproduction in whole or in part is permitted for any purpose of the U.S. Government

Preceding page blank

entirely with the first phase which was concluded in the summer of 1969.

2. SYSTEM DESIGN

Our data base is a closely intertwined mixture, mainly of Hebrew and Aramaic. Vowels are omitted, as is the practice in written Hebrew. In the texts, the "defective", the plene("full")and the "mixed" transcriptions (k'tiv chasser, male, and m'urav) are used. In the latter two, certain consonants are added in compensation for missing vowels, according to rules which are not entirely mechanical. Punctuation is generally missing. Acronyms and abbreviations with more than one possible interpretation are quite frequent.

The project is expected to be used mainly for research, preliminary studies for drafting new legislation, as well as for assisting the courts and lawyers. The potential users are not expected to require instant response. At any rate, we do not impose an a-priori time limit for processing a search. Rather, the emphasis is on in-depth searches, producing as high a recall as practically feasible. However, the system can process a number of searches simultaneously.

Our system design philosophy is based on the thesis advanced in (6), namely the establishment of a truly optimal sharing of responsibilities between man and machine, and the elimination of tasks done poorly by both. Among the latter we mention abstracting and indexing. Some advances have been made recently in automatic classification and indexing, including open-ended dynamic updatings in the form of a dialogue between the abstractor and the machine, improving consistency of indexing (1). But any conceivable abstracting method is inherently associated with some loss and distortion of the given information, since it is done today with a certain viewpoint by a given person, to be used tomorrow by a second person with a different and even unexpected viewpoint. Besides, abstracting processes are expensive and time consuming, whether performed by man or machine.

In our case there are further complications associated with indexing: The language of the responsa is often difficult to understand and is open to different interpretations. Any manual abstracting, which is normally done by beginners rather than by experts, forces an a-priori interpretation, which is of course undesirable. Also, the lack of vowels and the highly inflected and derivational nature of the language combine to render just about every word homographic. Thus, automatic indexing of acceptable quality of large Hebrew texts is at present as prohibitive as the manual approach.

These are the types of considerations which led us to the clear-cut conclusion that the only present-day method which might possibly be adapted to solve our problems is the full-text method. Its attractiveness is that it replaces a priori indexing by an ad hoc sifting method which can be optimized and tailored to the needs of the individual user and to his interests at any given time. In addition, it opens up the possibility of conducting searches on the basis of similarity of factual background, rather than solely on the legal principles involved, and it enables conducting linguistic researches, such as language development over a long period of time.

3. THE CONVENTIONAL FULL TEXT METHOD

We start with a few definitions:

A letter is any of the following symbols: the letters of the given alphabet, and some additional symbols. (In our case: the 22 letters of the Hebrew alphabet, the five "terminating letters" (see Appendix I), apostrophe(1), quotation marks(2).) Any other symbol is called a space.

A form is an ordered string of letters, preceded and followed by a space. The collection of all forms in a given text T is thus a set in the usual mathematical sense of the word.

A word is an appearance of a form in the text. Thus, the appearances or realizations of the forms of T , counting multiplicities, are the words of T . For example, the sentence "of the people, by the people, for the people" contains five forms and nine words.

In the conventional full text method, the entire text is entered in the machine. Once it is there, no separation of the text words into 2 sets, relevant and irrelevant, is undertaken by using global considerations as in automatic indexing. Rather, the machine performs such a separation by comparing each text word locally with a list of common forms given a priori. In this process, a concordance is produced and stored internally, giving for each non-common word its coordinates or text locaters, typically consisting of the document number, number of paragraph counted from the beginning of the document, number of sentence counted from the beginning of the document, number of word counted from the beginning of the sentence, and line type; that is, a code number specifying whether the word appears in a line of text, a line of an annotation, of a heading, etc. A document here means some predefined logical unit of law such as a statutory section or a case. Storage of the exact coordinates of each non-common word enables the definition of metrical

---

(1) which is placed above the last letter of an abbreviated word, indicating it being an abbreviation.

(2) which are placed between the last two letters of an acronym or vernacular to indicate it being an acronym or vernacular.

operators, whose iterative use permits citing all documents and only all documents containing given strings or classes of words within specified distances of other strings or classes. For more information about the method and references, see (6).

4. PREEDITING

Only the main functions of the preediting process are outlined below. The more technical details are omitted.

A fundamental rule of our preediting process is that not even a single letter of the original text be omitted or replaced. Additions and some text corrections are made, but only in a form that identifies them as such.

In order to enable the application of the "Sentence" and "Sentences" metrical operators (see Section 7.1), and for facilitating the understanding of short passages of text such as appear in KWIC indexes, the text is punctuated: Commas, semicolons, periods, question marks and exclamation marks are inserted by the preeditor as he scans the text. The sentence delimiters are the latter three punctuation marks, and the general guideline for punctuating is to generate short sentences.

The other relatively large preediting task consists of flagging some short text passages with certain classification numbers, for convenient printout of classified lists of textual material . About fifteen of these are for designating citations of other texts, such as the Bible, Talmud, G'onim, etc. The remaining flags (about 30), designate special topics, such as place names, personalities, realia, etc. Not every occurrence of these subjects is flagged. Thus, a citation which includes a reference is not normally flagged, since the reference enables the citation's retrieval without a flag(1). Similarly, only one occurrence of a given coin's name need be flagged, since a search by the appropriate flag number will retrieve the coin's name, and a second search will retrieve all occurrences of the name and its grammatical variants.

The interpretation of the non-standard acronyms and abbreviations is supplied and added to the text using special marks advising that an addition has been made.

The original serial numbering of the responsa is maintained by the computer. The pages and the lines within each page are numbered consecutively. Each line is punched on one (in some cases: two) punch-cards, together with the line numbers, which serve for later diagnostic purposes.

The output for an average preeditor is about 5,500 text words per workday. Plans for a possible omission of the punctuation process and other preediting processes will probably be considered some time in the future, aimed at speeding up the preediting. The present preediting is, however, still about twelve times faster than the manual indexing done currently by the team of indexers of the Institute of Research in Jewish Law in the project mentioned in Section 1.1 .

5. GRAMMATICAL PROBLEMS AND SOLUTIONS

5.1 The problems.

Hebrew is not just a different language than English; it is a language of an altogether different kind. Its highly inflected and derivational nature poses some serious and unique problems in every project of automatic Hebrew text processing, be it simple or sophisticated. Thus, even the automatic generation of concordances of a given text, which is quite a simple task for English texts, is very problematic for Hebrew texts. The essential point here is that a mere alphabetical listing of all forms of a Hebrew text —with or without context and reference — is absolutely unusable from an information retrieval point of view. Suppose, for example, that a user wants to retrieve all passages of a given text dealing with computers and computing. Ignoring for the time being the synonymity problem, he has to locate (or ask the computer to locate), among other terms, all appearances of the different variants of the verb "compute" and the noun "computer". These are: compute, computes, computed, computing; and computer, computers, computer's, computers', respectively. (In rare writings, the "e" in the four nouns is replaced by an "o", and those forms may also have to be added. Perhaps also all variants of the verb "computerize" have to be added.)

Note that all these forms have the same beginning kernel ("comput") and thus appear in proximity of one another (often following each other) in any alphabetically arranged word list of the given text. Thus the references of these twelve variants can be located immediately. They can be used either directly in a manual search or as input for a mechanized search.

It may be noted that this example is rather typical, because in general (excluding the irregular cases), the plural and the possessive forms of a noun, and the conjugated forms of a verb are constructed by changing (usually: adding, deleting) two or three of the last letters of the given form.

---

(1) Throughout this paper we use "reference" to designate the source location of a quoted passage, called the "citation".

Turning now to the Hebrew version of the above search, we first note that vowels are normally suppressed in printed Hebrew. Both the noun MHSB (machshev - computer) and the verb-root HSB (chashov - think) — which, in one of its seven grammatical structures known as modes (binyanim), means to "compute" — may appear in hundreds of variants; taking into account the plene transcription (k'tiv male) variants, even thousands of variants, which are not only different, but may also have different leading letters! In this specific example, there are up to twelve different leading letters. The following lists just one variant of each of these twelve sets: AHSB (I shall compute), BMHSB (in the computer), HMHSB (the computer), WKSHSBNW (and when we computed), HSWBNW (our computation), YHSB (he will compute), KSHSBTY (when I computed), LMHSB (to the computer), MHMHSBYM (from the computers), NHSB (we shall compute), SHSBT (that you computed), THSB (you will compute).

It may be pointed out, parenthetically, that there are actually 1030 variants of the noun MHSB (taken in the singular only)- and 5820 variants of the verb HSB (in the mode pi'el only). This was found in mid-1970 by a computer program implementing our solution to the grammatical problems. If run today, there will be more variants. The reason is that our programs in general — not only the grammatical routines — keep changing as a response to more sophisticated user demands. For example, since the data base covers many centuries, we keep finding new non-standard types of grammatical derivations, which we incorporate from time to time into our grammatical synthesis programs to be discussed in the sequel.

Note that the root HSB appears in different locations in the above representatives of the 12 variants. In fact, the root HSB itself may be broken by the insertion of different letters in-between (see HSWBNW above). However, it is true that the three letters H, S, B must appear in this order in any grammatical variant. But this is by no means true for all verbs. To take an extreme (but not irregular) example, the plural third person - past, of the root YRH (yaroh — to teach or to direct) is HWRW (horu — taught or directed), which has just one letter (R) in common with the root! (The leading "H" in HWRW is a prefix which is altogether different from the trailing "H" of YRH, which belongs to the root.)

By now it should be clear that the variants of a single Hebrew keyword are scattered all over any alphabetical list of the forms in the text. Locating them — which is essential both for manual and mechanical searches — amounts to scanning more than half of the given form-list, a clearly impossible task, especially since the form-list of Hebrew texts is much larger than its English counterpart (see Table I).

We note here briefly that more problems are superimposed upon those described above. They include the following:

(i) The suppression of vowels induces many homographs — words with the same spelling but normally quite unrelated meanings. For example, if we were to suppress vowels in English, the ordered string hll would induce homographs such as hall, hell, hello, hill, hilly, hull.

Most short words are particularly homographic, especially the so-called "common forms". In full text projects conducted in English, the suppression of about 120 common forms reduces the data base by about 50%. We cannot utilize this size reduction device, because of the many "non-common" meanings that our common forms possess. It should be pointed out, parenthetically, that this is not a total disadvantage. The common words, always described as "not containing any information", do, in fact, carry information which it is sometimes important to retain. For example, let C be any concept, S a search aimed at retrieving all documents which contain a formal definition of C . Then S can be formulated by the requirement that the name of C and its grammatical variants must be followed in the same sentence by one of the members of a set T , where T = {is, means, meaning, defines, defined, construes, includes} and the like. But if "is" is a common word, it cannot be included in this list, with a possible loss of relevant documents. It may be noted that such definitions are often useful. Indeed, in (11, pp. 17-18), a special macro instruction is described for retrieving all instances of a specific definition.

(ii) The plene transcription and its numerous variations and shades give rise to new classes of homographs.

(iii) The mixture of Aramaic and Hebrew forms, and the application of Aramaic grammatical rules to Hebrew forms and vice versa.

(iv) The mixture, in rabbinical prose, of several different "layers" of Hebrew language (biblical, talmudic, etc.), each having its own grammatical rules and peculiarities.

(v) The large number of acronyms and abbreviations, both standard and non-standard, that appear in the texts. In some passages, more than 50% of the text words are acronyms and abbreviations.

5.2 Inflectional and derivational patterns of Hebrew.

Before discussing possible solutions to the grammatical problems described, the inflectional and derivational patterns of Hebrew will be

TABLE I: WORD LENGTH DISTRIBUTIONS

Number n of letters per word	A. The first 197 responsa of Rivash: Number of forms with n letters	A. The first 197 responsa of Rivash: Number of words with n letters	B. All 518 responsa of Rivash: Number of forms with n letters	B. All 518 responsa of Rivash: Number of words with n letters	C. The total data base as of summer 1970: Number of forms with n letters	C. The total data base as of summer 1970: Number of words with n letters
1	20	182	21	273	22	659
2	236	20255	267	66648	285	161996
3	1639	39694	2322	135435	2824	309471
4	5461	45816	9445	157450	12910	362837
5	8228	34100	16043	114796	24372	270080
6	6270	17328	13699	57490	23105	133650
7	2819	6011	6764	20000	12148	45623
8	789	1575	2029	5449	3918	11802
9	201	332	551	1109	1070	2568
10	46	61	121	179	230	471
11	10	12	20	25	52	88
12	2	5	4	7	14	21
13	3	5	2	2	5	8
14	1	1	0	0	0	0
15	0	0	1	1	1	1
Total:	25725	165377	51289	558864	80956	1299275

explored briefly.

(i) The great majority of nouns and adjectives may have up to four different singular/plural masculine/feminine forms. These are derived from the singular/masculine form by adjoining to it certain suffixes. The derivation may be preceded by the deletion of some of the last letters of the singular form. Some of these forms undergo morphological changes when they appear in the construct state (smichut).

(ii) To most nominal forms, a variety of prepositions, consisting of one to four (in special cases: five) letters, may be added as prefixes, e.g., B (in), K (as), L (to), M (from), Š (that), KŠ (when), ŠB (that is in), MŠ (from the), KŠH (when the), W (and), the last of which may be prefixed to all others too.

(iii) Suffixes indicating the possessive forms (mine, yours, etc.) may be adjoined to most nouns, as well as to some adjectives. These suffixes have different forms according to the number and gender of the noun to which they are adjoined.

All these potential additions may occur in most (but not all) of the possible Boolean combinations. For example, one may add to the feminine form in a construct state prepositions and pronouns (a well-known example is the Biblical HKCĀKTH (Genesis 18:21), meaning: is it as her cry....?).

(iv) A verb-root (a stem of three — sometimes four — letters, traditionally the infinitive form of the verb) may be conjugated in up to seven modes, four tenses, ten persons and an infinitive form. In the three "active" modes, further suffixes may be attached, indicating the causative: me, you, him, etc. These may alter the endings of the forms they are adjoined to.

Finally, some of the prepositions mentioned in connection with nominal forms may also be prefixed to the verbal forms: B (and), Š (that), KŠ (when), and many more.

All this clearly indicates that in every automatic processing of Hebrew texts, a minimum amount of grammatical processing — automatic, manual or a combination of both — must be performed, if the results aimed at are not to be too trivial.

5.3. The adopted solution.

For English texts, manual variants of an "analytical approach" were advocated by some authors (5, 10, 16) as a solution to some of the grammatical problems. It consists essentially of manually grouping together all grammatical variants of a word under a Standard-Dictionary-Entry (SDE) form, followed by generating an alphabetically ordered index to be supplied to the user (as a manual guide) or to the machine (for automatic processing).

Some difficulties of this approach were pointed out in (6). For the Hebrew language the sheer bulk of the forms renders such an approach even more prohibitive (compare the statistics of Table I with Horty's project, where a corpus of about 6.25 million words was found to contain only 23,000 forms (6)).

On the other hand, an automatic construction of such a grammatical index — which can be done for English language texts — requires very delicate grammatical analysis routines and carefully prepared dictionaries, both painstaking and time-consuming tasks. (The feasibility of this approach was, nevertheless, demonstrated in a project carried out in 1964 (2,3), where a package of programs was written for and run on a Philco 2000 computer, which listed, for each given word, all its potential SDE's and only these.)

To bypass the time-consuming manual tasks implied by these solutions, we adopted, for the time being, a practical and somewhat more elegant solution, namely a synthetical approach. It is the "inverse" of the analytical approach. In this approach, the user gives the computer a "Standard Form" (SF) of the words he is interested in, together with some information about the type of grammatical variants he is interested in examining. The computer then generates all the variants of the desired types of the given SF, and searches the file for the appearance of all these variants. As may be expected, only a small number of all possible variants actually appears in the file (this number was about 20 on the average for the file whose statistics appear in Table IA).

This list of variants found in the text, with their frequencies, which constitutes a local frequency list, is given to the user, who decides which to discard and which to incorporate in the subsequent retrieval phases. This step is very desirable, since it is not uncommon that a certain variant of an SF appears in the text, but in an absolutely unrelated meaning. Such an occurrence can often be identified, assuming some knowledge of the nature of the text. To take an actual example: when asked for the variants of ŠD (shed — devil) appearing in the text, the computer produced, among other forms, ŠDY (shedi — my devil); however, even from a superficial knowledge of the text (and without any KWIC), one can infer that this form actually appears in the text in the meaning of "my field" (or perhaps: "the fields of..."), and thus can be eliminated from the subsequent retrieval phases. In cases where no clear-cut decision can be reached, the variant is retained, and other methods are resorted to (see "Compact KWIC" in Section 6).

The grammatical details of the synthesis algorithm may be reported on elsewhere; only the essential features are given here. For defining an appropriate SF for Hebrew forms, we were guided by three objectives: simplicity of the definition (so that even a layman in Hebrew grammar can easily determine the SF of any given form); comprehensiveness (i.e., that every grammatical variant of the given SF will indeed be generated); and simplicity of implementation (no need of manually built large files).

Accordingly, the SF was selected to be the masculine singular non-construct state variant for a nominal form, and the past third-person singular masculine variant in the desired mode for a verb. This is quite a departure from traditional grammatical thought, where the three-letter root is the source for all variants. However, since the meaning of a verb may be quite different — even contrary — in different modes, an SF as defined above is a most natural choice for our situation.

The generation algorithm is based on an "accumulating pool" principle, which means that the process consists of several phases, the n^{th} phase being applied to the set of all variants generated in the $(n-1)^{th}$ phase, the first set being the given SF.

For implementation of the verb-conjugation, three "fundamental" tables were constructed, each giving a unique image of all prefix+stem+suffix patterns of each person in each tense. The stem is denoted by a variable and is redefined for each given SF by a special subroutine which takes into account the SF's mode and formal morphological pattern. In addition, there are about fifteen short lists totalling less than five hundred entries for handling exceptional cases.

6. PHILOSOPHY AND STRATEGY OF SEARCHES

The full text method described in Section 3 is only a framework. In Section 5, we saw that additional ideas are required to prevent its collapse when confronted with a truly complex data base. From prevention of collapse to high-quality performance, there is an additional gap, which we bridge by means of feedback, both negative and positive, in a variety of forms. Our feedback is applied within the process of a single search, rather than by means of a sequence of searches ultimately converging to high recall and precision values.

To begin with, an expert in the subject matter of the search topic spends some time (perhaps one day) in trying to trace statutes, responsa and other material relevant to the topic. This material, if found, will suggest keyphrases as well as citations relevant to the search topic.

All the words selected in this phase are then subjected to grammatical synthesis as described in the previous section. The end product of this phase is, as we saw, a local frequency list, specific for the search at hand. The searcher scans it and for each entry he selects one of three possibilities: accepting it for the final search formulation; rejecting it; admitting it to a phase of finer scrutiny, called Compact KWIC. The first and third of these possibilities are explained below.

The "final search formulation", or simply "search formulation" is the phase of final input to the machine, incorporating the results of all previous feedback phases. It consists of two parts: (i) The definition of classes of forms, where each class is normally composed of all grammatical variants of a given keyform, or of a union of such classes. (ii) The application of suitable metrical operators to these classes, producing the — loosely or stringently defined — strings or sets of keywords and their clusters, whose presence distinguishes the documents to be cited.

A list of the available metrical operators is given in Section 7.1.

A "compact KWIC" is a normal KWIC, except that the neighborhood about the central keyword is rather small, consisting of about two words on either side, and the reference is usually omitted. This enables displaying a few compact KWIC lines on a single line of a page.

A compact KWIC is printed out for certain keywords selected by the user from the local frequency list*. It serves several functions:

One of the problems with full-text systems is that some relevant phrases (= strings of words) are not thought of by the search framer, although the central keyword of the string is used by him. A case in point is the "post mortem" performed by Horty on the results of the tests of his full text system (6,9). In search No. 2, "Care and treatment of unwed mothers", a relevant statute was missed, because the phrase "lying-in-hospital" was omitted by the framer, although "hospital" was used.

Another difficulty is that a keyform thought of originally as being very characteristic of a given search, may turn out to be highly homographic, causing many false drops.

---

* In the current first implementation, the grammatical synthesis and compact KWIC are generated in the same run. Therefore, a compact KWIC can currently be obtained only for all variants of a given SF, or for none of them.

Of course, synonyms present a particularly vexing problem. For example, the above-mentioned post mortem revealed that in search No. 1, "Treatment of infants in hospitals", one relevant statute missed dealt with childbirth by an inmate of a penitentiary. In this context, the word "institution", or even "penitentiary" turns out to be synonymous to "hospital". Another relevant statute missed used the phrase "state institution" instead of "hospital". In search No. 9, "Right to counsel in administrative proceedings", a relevant statute was missed since it used the phrase "appointing authority" instead of the normal "administrative agency".

The compact KWIC practically elminates the first type of problem. Thus, a compact KWIC on "hospital" immediately draws the user's attention to the phrase "lying-in-hospital". Also the second type of problem is practically solved by the compact KWIC, which indicates that the central keyword is homographic. The last type of problem is much more difficult. The legal discussions in our text, however, often contain repetitions of the same idea or concept and its supporting arguments, each time expressed in slightly different language. A KWIC on a known keyform can thus actually lead the user to a synonomous expression.

A more radical solution to the last problem would be the construction of special thesauri. However, as the above examples show — e.g. "appointing authority" being surprisingly synonomous to "administrative agency", even in a language which is not heterogeneous — our much more difficult text would require a very complicated thesaurus indeed, expensive and time consuming to construct. We decided not to choose this way for the time being.

Another aspect of the compact KWIC is that it reveals something about the quality of the search results, which can be utilized to advantage by the user for designing his final search formulation.

An important aspect of our literature is its extensive system of references to earlier responsal literature and to relevant statutory material. We take full advantage of this fact by including in the final search formulation either references to known relevant material found during the initial search phase, and/or keyphrases contained in the cited passages themselves. Thus, an effective citation system — containing the conventional citation index capabilities as a special case — is imbedded in our full text retrieval system. In this connection we cite from (13, p. 241): "There appear to have been no attempts to combine the citation searching with normal indexing techniques to form a mixed-access method. To some extent, the techniques appear to be complementary, and therefore some improvement in performance might be expected." We combine citation searching with full text searching, which indeed leads to improved performance.

Now, the reference of the citation, which consists of about four words, can conveniently be retrieved via a compact KWIC. Thus, relatively early in the search process we will know something about the relevant citations. For example, the absence of an important reference suggests that the desired topic may not be dealt with in the given text! If it appears in large numbers, on the other hand, we may deduce that the citation is referred to in connection with more than one topic. The user can then require the presence of the reference in conjunction with a collection of some of the basic keywords of the cited passage.

Similarly, the presence and absence of the context of other central keywords in the compact KWIC, leads the user towards a more intelligent final search formulation.

7. IMPLEMENTATION

Part of the contents of subsections 7.1 and 7.2 were already described in (4).

7.1. The metrical operators.

Every class of forms defined for the search formulation, as well as every class generated during the search execution — and we shall presently see that new classes are indeed generated during the search process — consists of the set of text locators (Section 3) of the words corresponding to the forms of the class. (For the sake of accuracy we point out that four of the operators generate sets of coordinates of documents rather than of words. The coordinates of a document consists of the author's code and the serial number of the document.) For example, if the user defines a class A = {people, peoples} , consisting of the two forms "people" and "peoples", and the first appears in six places in the text and the second in ten places, the computer generates a set of 16 coordinates of these words. The collection A can thus be viewed as a class of two forms (this was primarily our viewpoint prior to this section) or as a set of 16 coordinates of words (our present viewpoint).

The most important operators which we defined for operating on these classes are listed below. As used here, a document is synonomous to a responsum.

Range. The operator

$$\text{Range } (a_1,b_1;\ldots;a_n,b_n)$$

restricts the search to the subset of all documents within the intervals $(a_1,b_1],\ldots,(a_n,b_n]$.

Example: Range (10000,10200). The search is limited to the first 200 documents of author No. 1.

Define. The operator

Define $(A_1,\ldots,A_n;B)$

generates unions of given classes: $B \bigcup_{i=1}^{n} A_i$.

Words. The operator has the form

Words (A;m,n;B;C) ,

where A,B,C are sets, m,n integers. The set C is the collection of all the text locaters of B which are within the closed interval extending from m words after (if m is negative: before) a word of A to n words after (if n is negative: before) the same word of A , provided that the two words of A and B are in the same sentence.

In particular, the case $m = n = 1$ is used for constructing phrases. Another special case is the form

Words (A;m,n;B;-A) .

In this case, a text locater of A appearing within the specified interval about B will be deleted from A.

Examples: (i) Words (10;-2,3;11;20).

The set 20 will contain those text locaters of 11 which are within an interval of two words before to three words after a word of 10.

(ii) Words (15;1,1;16;18).

The set 18 will contain the text locaters of all those words of 16 which exactly follow a word of set 15. For example, if 15 contains the word "Federal" and 16 all grammatical variants of "Government", then 18 will contain, among other things, the text locaters of all the variants of "Government" which are preceded by "Federal". In this sense, the set 18 contains all text locaters of the phrase "Federal Government".

An iteration of this operator generates phrases of arbitrary length. For example, the phrase "information storage and retrieval" is generated by applying three such operators, each operating on the results of its predecessor.

(iii) Words (25;-1,1;26;-25).

In this example, a text locater of 25 appearing in the specified interval about a word of 26 will be deleted from 25. For example, in a search "The influence of weather conditions on the launching of satellites", in which the words "weather" and "satellite" will certainly play a role, the phrase "weather satellite" can be eliminated — if so desired — by the use of such a device.

Remark: The following six operators also have the same negation capability in the resulting set as the "Words" operator.

Sentence. The operator has the form

Sentence $(A_1,\ldots,A_n;B)$.

The set B consists of all those coordinates of words of A_n which are in the same sentence together with a word of each of the $A_i, i=1,\ldots,n-1$.

Sentences. The operator

Sentences (A;m,n;B;C)

is a binary operator like the operator "Words", and it has the same effect as the latter, except that its range is sentences instead of words. The two words, besides having to satisfy the specified sentence interval condition, are also required to be in the same paragraph.

Example: Sentences (8;2,5;10;15)

The set 15 will contain all text locaters of words of 10 which are within 2 to 5 sentences after a sentence containing a word of 8 , provided that both words are in the same paragraph.

Paragraph. The operator

Paragraph $(A_1,\ldots,A_n;B)$

has the same effect as the operator "Sentence", except that its range is one paragraph instead of one sentence.

The following operators each generate a set of document coordinates rather than word coordinates.

We shall say that the coordinate d of a document is imbedded in a set S of coordinates, if one of the following holds:

(i) S contains d ;
(ii) S contains a coordinate d_1 of a word w , and d_1 contains d (the word w is contained in document d).

Document. Its form is

Document $(A_1,\ldots,A_n;B)$,

and it has the same effect as the "Paragraph" operator, except that its range is a document rather than a paragraph: The set B will contain the coordinates of all documents p imbedded in each of the sets $A_1,\ldots,A_n$.

Document L. It has the form

Document L $(A_1,\ldots,A_n;k;B)$,

where k is a positive integer. The set B will contain the coordinates of all documents p imbedded in at least k of the sets $A_1,\ldots,A_n$.

Document Or. Its form is

Document Or $(A_1,\ldots,A_n;B_1,\ldots,B_m;C)$.

The set C will contain the coordinates of all documents p imbedded in at least one of $A_1,\ldots,A_n$ and in at least one of $B_1,\ldots,B_m$.

Cite. The operator has the form

Cite $(A_1,\ldots,A_n;-B_1,\ldots,-B_m;xy\ldots z)$.

This operator first generates the coordinates of all documents p imbedded in all of $A_1,\ldots,A_n$ and in none of $B_1,\ldots,B_m$. It then prints out these coordinates, followed by the mnemonic xy...z (optional). The mnemonic is used for indicating to the user the reason for the retrieval of the documents.

7.2 Block diagram of the system.

The system has been implemented on an IBM 360/50 with a 256 K byte memory, two magnetic tapes and a 2314 disk-pack of eight disks. The operating system is IBM's OS. All the software packages of the system —excluding standard service routines —were designed and written by us in Assembler language.

After reading in the text cards, the computer checks the continuity of the material, the presence of the correct number of parentheses and flags and various other logical conditions. Diagnostics and proofs are then printed out (see Fig. 1).

After correcting the errors, the computer generates a condensed text tape. At the same time it allocates to each text word its text locaters and its associated flags, if any, and generates records, each record containing a word with its text locaters and flags. These records are then sorted alphabetically by word, and within each word, chronologically. These sorted records form the basis for the dictionary and concordance, both of which are stored on disks.

The dictionary is an alphabetical listing of all the forms. Each record of the dictionary consists of one form, the number of documents it appears in, the total number of appearances of the form in the data base, and a link indicating the disk address of the first of the consecutively stored text locaters of this form. The list of all text locaters and associated flags constitutes the concordance.

The dictionary is an index sequential file. The concordance is a direct access file.

Updating the data base with new texts is —at this stage —a time-consuming task, since the dictionary, the concordance and their auxiliary files have to be copied during a merge. Unless more general list structures are used for the files — which usually implies more memory space and slower performance —the system cannot be updated on a real-time basis. At present, there is no real need to have this capability.

The text tape enables printing out the text with a numbering of the documents, the paragraphs and the sentences. This facilitates locating given passages.

For formulating a search, the user starts by defining classes of forms, as discussed in Section 6. He also assigns a number to each class.

For running one or more searches simultaneously, the computer undergoes two phases. In the first phase, the computer locates all the coordinates of each form in the given classes, by accessing the concordance via the dictionary. All these text locaters are extracted from the concordance, and for each of them the computer generates a record containing the coordinates, search number and class number (see Fig. 2).

These records are sorted by search number, and, within each search, by the chronological order of the text locaters. There is no sorting according to class number. At the end of this sort, the records are thus arranged chronologically within each search, and the forms giving rise to the records are not kept. They are of no further importance.

In the second phase, the metrical operators operate on these sorted records. For each search, the computer allocates a two-word stack for storing two text locaters of the records of each class, for a total of up to 100 classes.

The computer then scans the sorted records, clearing the stacks each time the document number of the scanned text locaters changes (i.e., increases) (see Fig. 3). The computer examines each record at its turn. If it is outside the range, it is discarded. Otherwise its coordinates are stored in the stack reserved for the class it belongs to. Thus the stack normally contains the text locaters of the two last scanned records. The computer now checks if any of the operator conditions is satisfied. If it is, the coordinates of the last class that fulfilled the condition are transferred to the new class defined by the operator. This new class is also assigned a two-word stack, and is henceforth treated as the previously defined classes. This is the case if the resulting class number is positive. If it is negative, the top word of the class' stack is deleted.

After the last record of a document is scanned (i.e., when the first record with a new document number is sensed), the computer checks

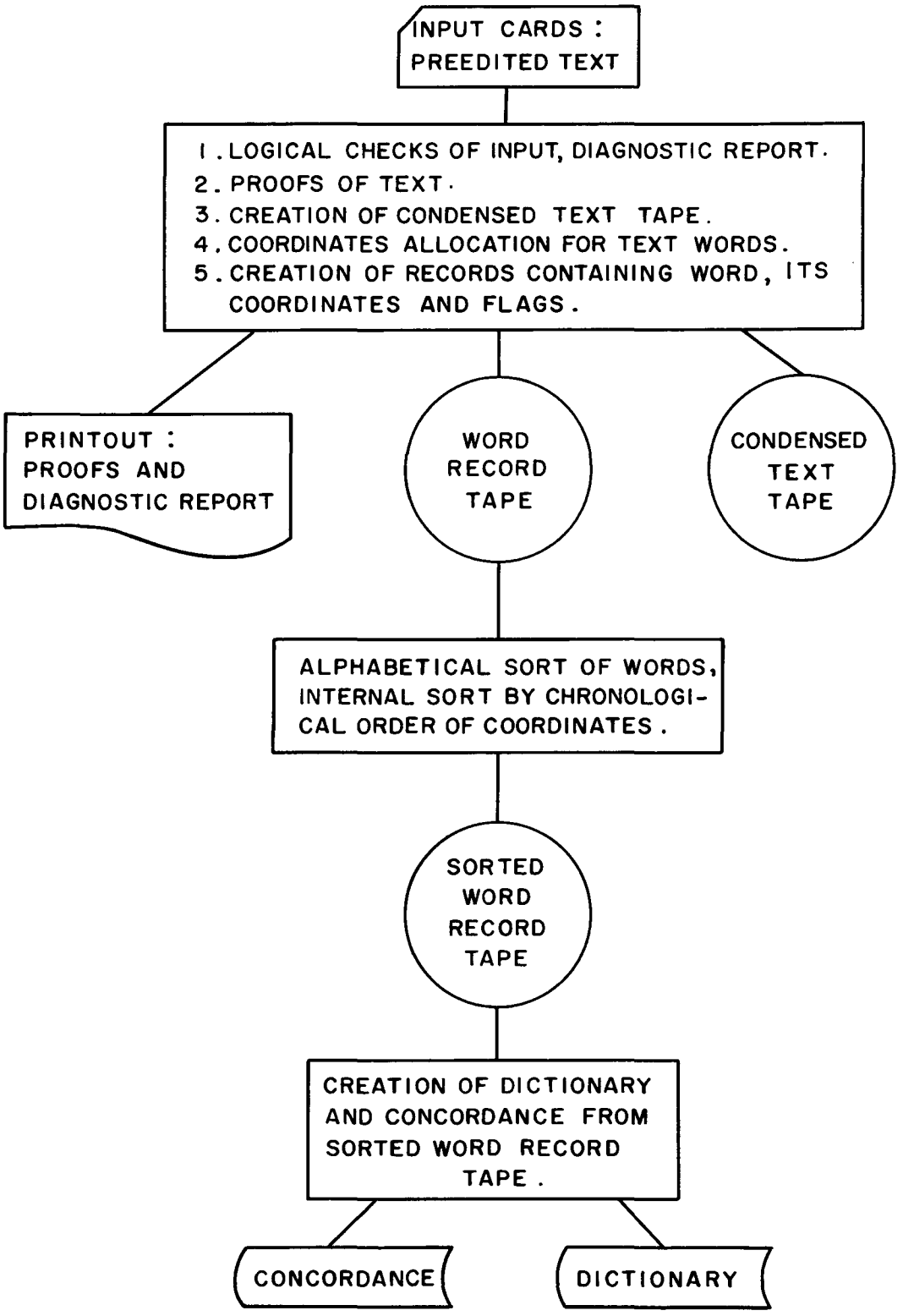

INPUT CARDS : PREEDITED TEXT
1. LOGICAL CHECKS OF INPUT, DIAGNOSTIC REPORT.
2. PROOFS OF TEXT.
3. CREATION OF CONDENSED TEXT TAPE.
4. COORDINATES ALLOCATION FOR TEXT WORDS.
5. CREATION OF RECORDS CONTAINING WORD, ITS COORDINATES AND FLAGS.
PRINTOUT : PROOFS AND DIAGNOSTIC REPORT
WORD RECORD TAPE
CONDENSED TEXT TAPE
ALPHABETICAL SORT OF WORDS, INTERNAL SORT BY CHRONOLOGICAL ORDER OF COORDINATES.
SORTED WORD RECORD TAPE
CREATION OF DICTIONARY AND CONCORDANCE FROM SORTED WORD RECORD TAPE.
CONCORDANCE
DICTIONARY

FIG. 1 : FILE CONSTRUCTION DIAGRAM.

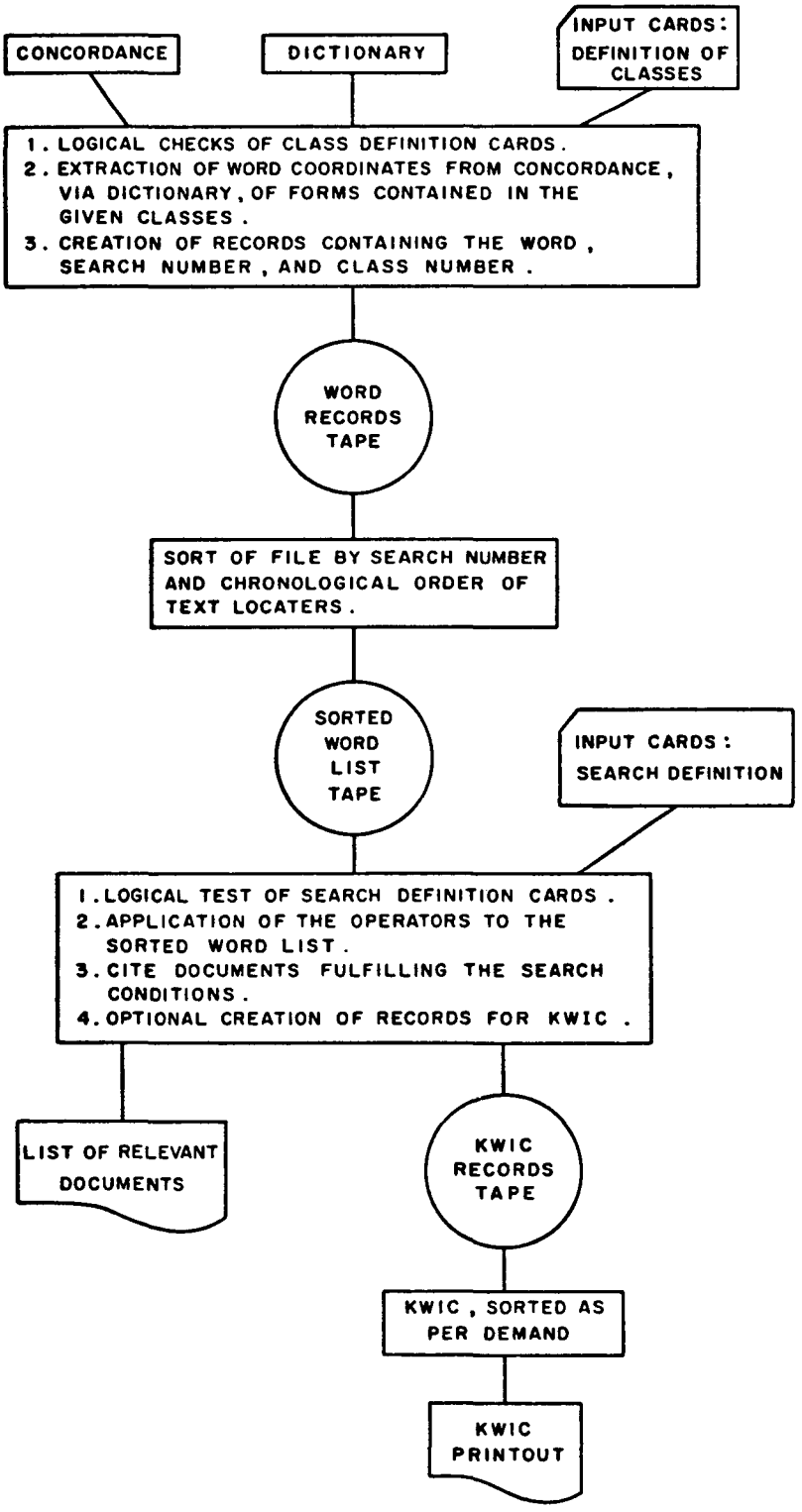

CONCORDANCE
DICTIONARY
INPUT CARDS: DEFINITION OF CLASSES
1. LOGICAL CHECKS OF CLASS DEFINITION CARDS.
2. EXTRACTION OF WORD COORDINATES FROM CONCORDANCE, VIA DICTIONARY, OF FORMS CONTAINED IN THE GIVEN CLASSES.
3. CREATION OF RECORDS CONTAINING THE WORD, SEARCH NUMBER, AND CLASS NUMBER.
WORD RECORDS TAPE
SORT OF FILE BY SEARCH NUMBER AND CHRONOLOGICAL ORDER OF TEXT LOCATERS.
SORTED WORD LIST TAPE
INPUT CARDS: SEARCH DEFINITION
1. LOGICAL TEST OF SEARCH DEFINITION CARDS.
2. APPLICATION OF THE OPERATORS TO THE SORTED WORD LIST.
3. CITE DOCUMENTS FULFILLING THE SEARCH CONDITIONS.
4. OPTIONAL CREATION OF RECORDS FOR KWIC.
LIST OF RELEVANT DOCUMENTS
KWIC RECORDS TAPE
KWIC, SORTED AS PER DEMAND
KWIC PRINTOUT

FIG. 2: DIAGRAM OF SEARCH PROCESS.

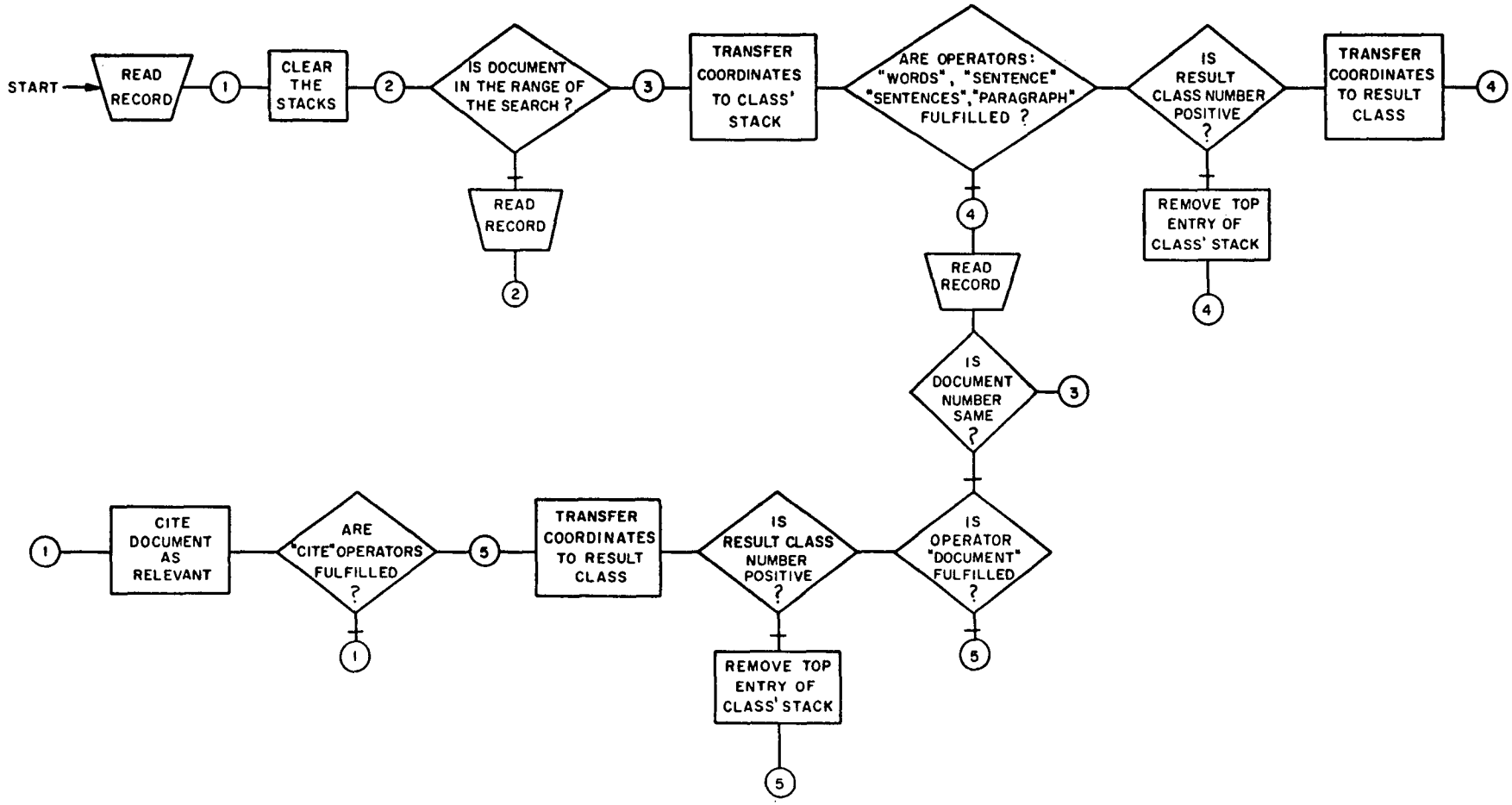


FIG. 3: THE FUNCTION OF THE METRICAL OPERATORS.

the "Document" and "Cite" operators. If the conditions of any of the "Cite" operators are satisfied, the document is cited by printing out its serial number. Optionally, a KWIC of the sentence(s) carrying information causing retrieval is also printed out.

7.3 Sample search.

a. Topic: Recovery of Expenditures in Breach of Promise. Is one who breaches a promise to marry liable for the expenditures made by the fiance(e)?

Two distinct types of recoveries are considered: (i). Gifts of property from one party to the other. (ii). Purely monetary expenses incurred in the performance of those acts demanded by custom, such as festive meals, monetary gifts to servants, contributions to charity, and travel expenses for visiting. The legal question dealt with in the first category is usually ownership of the property (return of the gift), whereas the second, since it deals with repayment of funds laid out, is a part of the law of torts, more precisely, indirect damage, as the defendant committed no direct physical act of injury; indeed, the damage was committed by the injured party upon himself. In the relevant legal literature, as in real life, both problems are usually intertwined.

b. Sources. The primary source for the legal discussion of the expenses' topic is a case found in the Code of Maimonides, where the groom-to-be made the customary party for his friends and distributed gifts, and the bride-to-be then broke the engagement. Maimonides, following precedent, finds against the bride in such cases and requires repayment of all expenses. Ravad (Rabbi Abraham ben David, 12th century, Provence), Maimonides' arch critic, disagrees, basing himself upon a Talmudic decision wherein one who sells defective seed is liable only for the cost of the seed, but not for the expenses incurred by the buyer in unsuccessfully planting such seed. Thus we see —maintains Ravad —that if the immediate act of injury is performed by the plaintiff himself, but not by the original causer of the loss, the indirect act of damage involved cannot generate liability. The commentators to Maimonides' Code explain Maimonides' position as differentiating between an act of business, such as the seed transaction, and an act dictated by local custom, as in our case of engagement. In the latter, one is liable even for what would normally be considered an indirect cause of damages. The later codes side with Maimonides, and such is the accepted practice in the courts (see Shulchan Aruch, Even HaEzer, Section 50, paragraph 3, and Aruch HaShulchan, loc. cit.).

c. General search strategy. Following is a list of the keywords and keyphrases to be used in the mechanized search.

Keyword combinations	Explanations
1. (Tractate) Baba Batra, (Folio #) 92	Talmudic source germane to our topic
2. (vegetable) seeds	Subject of discussion in the text of the above Talmudic cite
3. Code of Maimonides, Zechiah U'Matanah, Chapter 6.	Reference to code dealing with the problem
4. (Prevalent) custom, feast, breach, cause (of damage)	Terms used by Maimonides in #3
5. Even HaEzer (Sect.) 50	Reference to later code
6. (Marriage) agreement, gifts, expenditures, indirect cause (of damage)	Other terms germane to the topic, taken from the literature

d. Definition of classes. The classes below are defined in the order of c. sbove. Each class contains, in actuality, all grammatical variants — of the given single entry —which occur in the data base.

Although an attempt was made to preserve as much as possible the flavor of the original Hebrew search formulation, some minor variations were introduced to provide meaningful illustrations to the English reader.

10 = Baba	23 = feast
11 = Batra	24 = breach
13 = (Folio#)92	25 = gifts
15 = (vegetable) seeds	27 = Even
16 = Maimonides	28 = HaEzer
17 = Zechiah	30 = (Section)50
18 = U'Matanah	32 = (Marriage) agreement
20 = Chapter 6	33 = expenditures
22 = (Prevalent) custom	35 = indirect cause (of damage)

e. Search formulation. A document quoting a source, frequently brings only the reference or the citation, not both. When we want to retrieve such a document, we thus designate it as relevant if it contains either the reference or the citation. This of course increases the recall at the expense of the precision (but see Section 9).

Operator	Explanations
*Words (10;1,1;11;12)	Baba Batra
*Sentence (12,13;14)	Baba Batra 92
Words (17;1,1;18;19)	Zechiah U'Matanah
Paragraph (16,19,20;21)	Reference to Maimonides
Document L (22,23,24,25, 32;3;26)	3/5 keywords used by Maimonides
*Words (27;1,1;28;29)	Even HaEzer
*Sentence (29,30;31)	Even HaEzer 50
Document (32,33;34)	Agreements, expenditures
*Document (35,15;36)	Seeds, Indirect damage
Document L (24,25,32,33; 2;37)	Breach, agreement, etc.
*Cite (14;B.B.92)	
Cite (21; Maim Zum)	cited 2 relevant, 1 irrelevant doc.
Cite (26;3/5)	cited 4 relevant, 8 irrelevant doc.
*Cite (31; E.H. 50)	
Cite (34;Agr/Exp)	cited 1 relevant, 9 irrelevant doc.
*Cite (36;Seed/Dam)	
Cite (37;Br/Agr)	cited 4 relevant, 23 irrelevant doc.

The lines with asterisks designate operators which were included in the original search plan but not in the final search formulation, since the first phase, in particular the Compact KWIC, revealed that the keywords they operate on do not exist in the data base. We retained them here for illustrative purposes only.

f. Results. The valuable knowledge gained by the Compact KWIC indicated that the expenses topic could only be mentioned in the text en passant. Therefore a rather broad final search was formulated with those few keywords left. It was indeed found that the expenses topic was mentioned only in a single sentence in a data base of half a million words. Four documents were found to be relevant to the gifts' topic. Due to the broad search frame, 23 false drops were obtained, and quickly disposed of by scanning a KWIC of the search results.

8. INITIAL TESTS AND THROUGHPUT TIMES

The first responsa read into the computer were those of Rabbi Isaac bar Sheshet (referred to in the sequel by the mnemonic "Rivash", an acronym of his name in Hebrew, or simply by "R"), who lived in the second half of the 14th century, first in Spain, and after the first deportation of Jews (1391), in North Africa.

At the first stage, about one-third of the book was read in, comprising 197 responsa, which constitutes the data base of Part A of Table I. Ten searches were run on it. After that, the balance of the book was read in and merged with the first part, comprising the data base in Part B of Table I. The first ten searches were then rerun on this expanded data base with slight changes, and additional tests were run. The results are depicted in Table II. Results of two of these searches were already reported on elsewhere (14, 15).

Each of these searches was run only once. But our very first search, which was exploratory in nature, was run several times, to test various techniques. It was the same topic as search No. 1 of Table II, but run on the first third of R only. A "postmortem" of the very first run of this exploratory search revealed that two relevant documents were missed, because we neglected to incorporate into the search formulation a certain two-word phrase of a keyform, although the keyform itself was used by us in the search formulation — but only in combination with other keyforms. By the way, this failure induced the idea of the Compact KWIC.

The most conspicuous result is of course the consistent complete recall. Whether we can retain this for a larger data base is a question we hope to find an answer to in the second phase of our experimental project. Meanwhile we have only demonstrated the success of the approach for small data bases.

A KWIC of those portions of each cited document which caused its retrieval is normally printed out with the results. It enables the user in most cases to decide at a glance whether the document is relevant or not. Therefore the precision figures, which were not too high, do not worry us too much for the time being.

It would be most natural to have our retrieval system imbedded in an interactive time-shared environment. Such a luxury was not available to us, however. We had to work in a normal batch mode. Thus, at least two runs per search had to be performed: one for the grammatical synthesis,

TABLE II: SUMMARY OF SEARCH RESULTS

Search No.		Data Base	No. Relevant Retrievals	No. Irrel. Retrievals	Recall %	Precision %
1	Occult, demons, spirits	R	7	16	100	30
2	Marranos of 1391	R/3	17	9	100	65
3	""	R	17	74	100	19
4	Intent on part of gift recipient as prerequisite for acquisition	R/3	No truly relevant material was found			
5	""	R	1	26	100	4
6	Preference in levirate marriage given to elder brother claiming chalitzah or to younger brother requesting yibum? (See Deut. 25,57).	R/3	2	1	100	67
7	""	R	5	2	100	71
8	Is seclusion after matchmaking sufficient for marriage?	R/3	3	7	100	30
9	""	R	3	10	100	23
10	Recovery of expenditures in breach of promise	R/3	3	9	100	25
11	""	R	4	23	100	15
12	Legality of act of acquisition performed without presence of witnesses	R	5	24	100	17
13	Testimony of physicians as to cause of death	R	2	3	100	40

TABLE II: SUMMARY OF SEARCH RESULTS (CONTINUED)

Search No.		Data Base	No. Relevant Retrievals	No. Irrel. Retrievals	Recall	Precision
14	May wife appoint agent to receive writ of divorce from husband's agent?	R	1	2	100	33
15	Can husband's usage of wife's property constitute proof of his ownership?	R	1	3	100	25
16	All locations of the Hebrew phrase mikol makom (nevertheless)	R	-	-	-	-
		Average:			100	34

REMARKS

R/3 and R are the data bases of Table I, made up of the third and the total of the Rivash text, respectively. In Searches No. 1 and 2, we retrieved the most comprehensive source list for Rivash known at present. In Search No. 4, five responsa were retrieved. One is close to the topic sought and three others also deal with the general topic of intent with respect to gifts. One is completely irrelevant. None was judged to be truly relevant. In Search No. 5, 17 of the 26 irrelevant responsa deal with the general topic of intent in gifts. A detailed description of Search No. 11 is given in Section 7.3 . In Search No. 13, one of the three irrelevant responsa nevertheless dealt with the general search topic. In the last search, the recall and precision were not computed. The client was apparently satisfied with the results.

which was combined with the Compact KWIC phase, and the final search. Sometimes the results of the first run suggested performing a supplementary run, i.e., subjecting additional keyforms to grammatical synthesis.

Each search took about four work-days to run to completion. This included the initial research of one day, the communication with the — at that time outside — computer, and the determination of the recall and precision figures by our team, with painstaking checks of the results. These checks included scanning huge KWIC printouts of all grammatical variants of all keywords taken individually — not in combination — and scanning simultaneously selected passages of the full text and a number of auxiliary books. (The bulk of these checks — such as scanning the huge KWIC — is obviously omitted in production runs on larger data bases.) The time of four days does not, however, include the check of the recall and precision performed by the outside experts, who were mainly the members of the Institute of Research in Jewish Law of the Hebrew University, headed by Professor M. Elon.

In an interactive mode of operation, this time can probably be reduced, possibly to a few hours, but some new ideas would have to be used for transforming the system into a "hot" real-time retrieval system with response times of minutes.

9. LATER INNOVATIONS

The second phase of the project was begun in the summer of 1969. It is expected to be completed in the summer of 1971. Three goals were set:

(i) Enlarging the database to approximately one and one-half million words.

(ii) Testing the system by running about one hundred searches.

(iii) Development, refinement and sophistication of the software.

As of summer 1970, the data base had been expanded to about 1,300,000 words, by adding the responsa of "Rosh" (Rabbi Asher ben Yechiel) and parts of the responsa of "Rashbah" (Rabbi Shlomo ben Aderet), both Spanish respondents of the 12th, 13th centuries. We are now in the process of adding responsa written in the 14th century in Germany, and responsa written in the 18th and at the beginning of the 20th century in Central Europe.

Enlarging the data base also presented us with the opportunity to study certain problems of handling large volumes of information. As a result, an updating program was written, which handles the correction of errors, additions and deletions.

The grammatical programs were also revised, with a two-fold aim: further delegating to the machine some of the tasks previously done by the user, and further developing the algorithms so as to come closer to the ideal of synthesizing all possible grammatical variants of a given keyword. In order to handle efficiently the large amount of variants thus generated, most of which will be rejected, a fast sophisticated method of searching was developed and is presently being implemented. The method eliminates the need for sorting the generated forms. Instead, it makes use of special densely packed core-memory text-dictionaries and indexes. The great majority of variants are killed off "on the fly", even before reaching the stage of being stored.

A new option for printing out the retrieved document numbers was devised, whereby the documents are sorted by the number of different "Cite" operator conditions fulfilled in each document, ranked in descending order. Scanning this graded list, with the associated "Cite"-mnemonics, allows the user to draw tentative conclusions as to the probable relevance of a given document. This is of special importance for a large data base or for large returns.

We are also currently studying the possibilities of imbedding our system in a more interactive environment, and using optical scanning equipment for data input.

ACKNOWLEDGEMENT

We welcome this opportunity to express our appreciation to the Institute of Research in Jewish Law at the Hebrew University and its Council, to the Weizmann Institute of Science and to Bar-Ilan University, for their valuable support and cooperation. The contribution of the Institute of Research in Jewish Law is especially acknowledged.

APPENDIX I: HEBREW TRANSLITERATION

Hebrew orthography consists of 22 letters (consonants). Five of them, when appearing at the end of a word, are designated by special symbols called "terminating letters". In addition, there are diacritical marks, which denote vowels, and, sometimes, the doubling of consonants. Moreover, in some letters the absence of a diacritical mark indicates it being one consonant (V, Ch, F, S), whereas the presence indicates it being another (B, K, P, T respectively). But it is standard and universal practice — except in some special texts — to omit these marks, both in ancient and in modern writings, it being assumed that the relevant information can be extracted uniquely from

the word's consonants and its context in the sentence (which is usually —but not always — true). Punctuation marks are normally absent from all ancient writings, but they are being used in modern writings.

To give as faithful a picture of the situation as practically feasible, we adopt here a one-to-one transliteration of Hebrew letters to English letters (see Table III). Each word will be written in up to three forms: the transliteration (in capital letters) followed by the pronunciation (underlined) and the translation. Examples: $\bar{S}$LWM (shalom — peace); TMYD (tamid — always).

TABLE III: HEBREW-ENGLISH TRANSLITERATION

א	A	ח	$\bar{H}$	ע	$\mathring{A}$
ב	B	ט	$\bar{T}$	פ	P
ג	G	י	Y	צ	C
ד	D	כ	K	ק	Q
ה	H	ל	L	ר	R
ו	W	מ	M	ש	$\bar{S}$
ז	Z	נ	N	ת	T
		ס	S		

REFERENCES

(1) BENNETT, J. On-line access to information: NSF as an aid to the indexer/cataloger. Amer. Documentation Vol. 20 (1969) pp. 213-220.

(2) CHOUEKA, Y. Automatic grammatical analysis of the Hebrew verb (in Hebrew). Proc. 2nd Nat. Conference Inform. Process. Assoc. Israel, Rehovot, 1966, pp. 49-67. English abstract: The Finite String Vol. 4 (1967).

(3) CHOUEKA, Y. and SHAPIRO, M. Machine analysis of Hebrew morphology: potentialities and achievements (in Hebrew). Leshonenu Vols. 27-28 (1964) pp. 354-372. English abstract: The Finite String Vol. 3 (1966).

(4) COHEN, M. Full text information retrieval (in Hebrew). M. Sc. Thesis submitted to Tel-Aviv University (1970).

(5) ELDRIDGE, W. B. and DENNIS, S. F. The computer as a tool for legal research. Law and Contemporary Problems Vol. 28 (1963) pp. 78-99.

(6) FRAENKEL, A. S. Legal information retrieval. Advances in Computers. Vol. 9 (1968) pp. 113-178.

(7) FRAENKEL, A. S. Modern retrieval methods applied to Jewish case law. Proc. First International Symposium Soc. Tech. Writers and Publishers, pp. R-1-1 — R-1-9, (December 1968), Tel-Aviv, Israel.

(8) FRAENKEL, A. S. The computer and information retrieval as auxiliary tools in Hebrew legal research (in Hebrew). Technical Report, Weizmann Institute of Science and Bar-Ilan University (1969).

(9) HORTY, J. F. Searching statutory law by computer, Interim Report No. 2. Health Law Center, Univ. of Pittsburgh, Pittsburgh, Pennsylvania (May 1962).

(10) KALIKOW, M. A long-range program for mechanized legal and patent searching centers. Modern Uses of Logic in Law (June 1962) pp. 78-86.

(11) NIBLETT, G. F. B. and PRICE, N. H. The status project: searching atomic energy law by computer. Research Group Report CLM-R101. United Kingdom Atomic Energy Authority, Culham Laboratory, Abingdon, Berkshire (1969).

(12) SCHREIBER, A. M. Computerized storage and retrieval of case law without indexing: the Hebrew responsa project, Law and Computer Technology (November 1969) pp. 14-22.

(13) SENKO, M. E. Information storage and retrieval systems. Advances in Information Systems Science. Vol. 2, pp. 229-281. J. T. Tou, editor, Plenum Press (1969).

(14) SLAE, M. Marranos' search in the Rivash responsa. Bar-Ilan Yearbook. Vols. 7-8 Hebrew section: pp. 397-419. English section pp. XLVII-LXIII (1969-70).

(15) SLAE, M. The occult in the Rivash responsa (in Hebrew). Shanah B'Shanah 5731, pp. 226-236. Heichal Shlomo, Jerusalem (1970).

(16) WILSON, R. A. Computer retrieval of case law. Southwestern Legal J. Vol. 16 (1962), pp. 409-438.

QUANTIFICATION IN QUERY SYSTEMS

J. L. Kuhns
Operating Systems, Inc., Encino, California

ABSTRACT

Questions which involve 'all', 'every', 'some', or the indefinite article, pose some peculiar problems when presented to a computerized question-answering system where ambiguities cannot be tolerated. These problems vary from the nature of the correct answer in special cases to the very admissibility of the question itself. To deal with these problems it is convenient to divide questions into two classes---extensional questions whose answers are to name things or truth values, intensional questions whose answers are to give meanings. This paper examines extensional questions. For these, the interpretative problems arising with 'all' and 'every' can be solved by introducing a new kind of quantification, extensional universal quantification, that has the meaning of 'all F' together with a secondary meaning that the class F is not empty. Formal rules for this quantification are given, and it is shown that the so-called definite formulas (which explicate permissible queries) are closed under the new operator.

KEY WORDS AND PHRASES

Data retrieval, first-order predicate logic, question-answering systems, relational data file, semantics.

I THE PROBLEM

Our purpose in this paper is to examine the role of quantification in computerized question answering systems. For illustrative purposes, let us consider our system to have a data base consisting of bibliographic and related information on publications and their authors.+ This has the advantage of providing concrete examples for the theoretical development. Thus, we pose the problem of processing questions such as

Did Carnap write a book? (1)

Is every publication of Reichenbach a book? (2)

Who wrote all the papers on inductive logic? (3)

These questions have a common feature ---they involve quantification. Question (1), through use of the indefinite article, involves the existential quantifier; i.e., the existence of an object of a certain kind is questioned. Questions (2) and (3) involve, as indicated by the words 'every' and 'all', respectively, the universal quantifier.++

But these three questions differ in a certain important aspect---they display, in a sense, three levels of "reasonableness" of query. Thus, question (1) is

---

+For a description of such a system see Levien and Maron [13, 14].

++Question (2) also contains an indefinite article and this flags the presence of an existential quantifier; but the combination of the identity relation 'is' with the indefinite description 'a book' is actually a paraphrase of the subject-predicate relation and can be directly eliminated. That is to say, question (2) has a logical equivalent 'Are all of the publications of Reichenbach books? .

Preceding page blank

undoubtably reasonable and unambiguous; but, for question (2), there are special cases in which we would be uncertain as to its proper answer. For example, in the case that there are no publications of Reichenbach, then is the answer to (2) 'yes' or 'no'? The third question is an "x-question" (to be explained below) and presents still another difficulty; namely, to what extent should we restrict the range of possible answers? To solve these problems requires us to carefully analyze the relation of quantification to our explication of "reasonable" question. We shall show that we can indeed give rules for bringing all three types of questions to the same level of acceptability for a computerized question-answering system.

II BACKGROUND NOTIONS

Two Kinds of Questions. Otto Jespersen, in his 1924 work "The Philosophy of Grammar," gives a rather straight-forward concept of questions which provides a good starting point for founding a theoretical basis for computerized question-answering. He distinguishes two broad categories of questions and, as a terminology, suggests the terms nexus-question and x-question. Questions (1) and (2) above are examples of the former, while (3) is an x-question.
Jespersen writes [9, p. 303]:

> "...in the former kind it is always a nexus the truth of which is called in question: the speaker wants to have his doubt resolved whether it is correct to connect this particular subject with this particular predicate. We may therefore call questions of this kind nexus-questions. In the other kind of questions we have an unknown 'quantity' exactly as in an algebraic equation; we may therefore use the well-known symbol x for the unknown and the term x-question for a question aiming at finding out what x stands for."

The same idea was developed, but with the advantage of logical notation, in a remark by Carnap in "The Logical Syntax of Language" [5, p.296]; and carried further by Reichenbach in a more elaborate discussion in "The Elements of Symbolic Logic" [15, pp. 339-342].+ The technique of both Carnap and Reichenbach is to formulate x-questions as "open sentential formulas" of the predicate calculus (i.e. symbolic translations which contain a free variable) and then to prefix these formulas with an interrogative operator '(?x)' which specifies the information requested. For example, the question 'Who wrote Waverley?' would be symbolized by using the two-place predicate 'W' (for 'wrote'), and an individual constant 'b' (for 'Waverley') to yield

$$(?x)(Wxb) \qquad (4)$$

Symbolic Questions and Value Sets. For simplicity we shall omit the interrogative operator and speak only of the sentential formula. This formula we call the symbolic question. It is not the question itself, but only the expression that must be processed to answer the question. The processing of questions by computer leads to special problems. Thus the acceptable answers must be

---

+Reichenbach gives evidence to show that the division into two kinds of questions is indeed general. For example, he points out that a why-question asks for an event. We should note that, in the same work, Reichenbach gives philosophical grounds for considering events to have "the physical existence of things, rather than the fictitious existence of situations or properties." Thus the classification of a why-question as an x-question seems further justified.

carefully stated. Consider, for example, the question 'Who is Scott?'. An acceptable answer would be a description of Scott, but not 'Scott'.+ Carnap raises the inverse problem in [6, pp. 71, 82]; he mentions the problem of giving an extension. Recall that an open sentential formula can be used to define a property or relation. This is done by prefixing the formula with the so-called λ-operator [4, p. 129]. Thus '(λx)(Wxb)' is a predicate expression denoting the property that x has when it stands in the relation W to b; i.e., in the present case, the property of being an author of Waverley. The things that have the property comprise the extension of the predicate. It is evident, therefore, that question answering deals rather closely with the problem of giving an extension. Consequently, we must shift the discussion to the metalinguistic level so that we can control the form of the answer. Thus a name is requested in (4), not a description. For this reason I have introduced the concept of value set. While the extension is on the object level, the value set is always an expression or a set of expressions. For a symbolic question with a free variable x, the value set is the set of expressions of a certain specified type which, when substituted in the symbolic question yield a "true" sentence. Similarly, for a symbolic question without free variables (Jespersen's nexus-question) the value set, or simply the value, is an expression giving the truth value.

The concept of intension [4, 6] is also useful in the theory of questions. Thus, the anomolous question 'Who is Scott?' mentioned above can be viewed as asking for an intension; an appropriate answer would be a "dictionary definition" of 'Scott'.+ Another example of an intensional question is 'What is a logician?'. If meaning postulates [6,pp.222f] were included in a file, then the meaning postulates concerning 'logician' could be given as an answer in lieu of other expressions giving intensions---such as dictionary definitions. An extensional-intensional analysis thus reveals a rather close relation between 'Who is a logician?' and 'What is a logician?'.

Note that the extension of a sentence can be taken to be its truth value[4, p. 39f; also 6, p. 25f]. Thus, there seems to be an alternate division of questions; namely, questions whose answers are to give extensions (truth values, individuals) and those whose answers are to give intensions (propositions, individual concepts, properties and relations).++

+It is possible to interpret this question in terms of events. Thus, the "worldline" of Scott is asked for, i.e., the events of a certain set. The description 'the author of Waverley' characterizes an individual with the same worldline as Scott. But we shall prefer another interpretation.

+For a completely different analysis of this question, see Åqvist [1, p. 98]. But I cannot regard his analysis as satisfactory; in fact, it seems that Åqvist improperly classes the question as of the same type as 'Who is the author of Waverley?'.

++Interestingly enough, the earlier Jespersen-Carnap-Reichenbach concept of questions seems more appropriate as a basis for computerized data retrieval than later work such as that of Belnap [2] or Harrah [8]. In work directly related to computer implementation,Bobrow [3] has two categories of questions but no details of definition are worked out and he refers to the answers to x-questions as objects! This is criticized by Kasher [10, p. 310f], who states that Harrah's concept is "superior." It seems to me that Harrah's explication of questions as declarative sentences of a specified form lacks a good match with (what Carnap [6, p.8] would call) the explicandum. None of these authors refer to Jespersen, Carnap, or Reichenbach.

Data Bases, Files, and Dictionaries. For computerized question-answering we must have a stock of "true" sentences from which value sets can be calculated. In the simplified model of question-answering to be considered by us, this stock is called the file and it consists of a set of atomic sentences; i.e., predicate expressions followed by an appropriate number of individual constants. The set of permissable substitutions is called the dictionary and completely inventories our universe of discourse. The file, together with the dictionary, comprises the data base.+ For example, let a dictionary consist of 'a' for 'Scott', 'b' for 'Waverley', 'c' for 'Carnap', 'd' for 'Meaning and Necessity', 'e' for 'Reichenbach', 'f' for 'Elements of Symbolic Logic', 'g' for 'What is Probability?'. Let our file be built from a two-place predicate 'W' to denote the relation of authorship (in a broad sense), a one-place predicate 'B' to denote the property of being a book, and a one-place predicate 'I' to denote the property (of a publication) when it concerns inductive logic. Then a file which matches the true facts of our world is given by the following atomic sentences: 'Wab', 'Wcd', 'Wef', 'Wcg', 'Bb', 'Bd', 'Bf', 'Ig'. It is convenient to add to every file a special sentence 'T' which is always true. This completes our example data base; we will refer to it as D_o.

The value set of a question whose symbolic question is the sentential formula s will be denoted by'ω(s)'. Thus, the value set corresponding to question (4) is ω('Wxb'), and for the example data base D_o, we have ω('Wxb') = {'a'} .+ The corresponding calculation for the question 'Did Scott write Waverley?' gives ω('Wab') = {'1'}, where '1' is a symbol denoting truth. For simplicity, we shall write this last result as

$$\omega(\text{'Wab'}) = 1$$

Atomic sentences, such as 'Wab', are therefore evaluated by a direct search of the file; the value is'1' if the sentence is found, otherwise the value is'0'. Thus the file is considered to give complete extensions of the primitive predicates.

Definite Formulas. Consider the effect of adding a new entry to the dictionary of D_b. Let this entry be the symbol '*', which is a special "logical test name." Formulas are now divided into two classes: those whose value sets are changed by the addition of '*', and those whose value sets are unchanged. For example, we have seen that ω('Wxb') = {'a'}. The new calculation, with the addition of '*', allows one more substitution; but the result, which we call ω^*('Wxb'), remains the same. Thus,

$$\omega(\text{'Wxb'}) = \omega^*(\text{'Wxb'})$$

It is easy to show that this invariance will be the case for any atomic formula, regardless of the composition of the file, i.e., the truth or falsehood of any atomic sentence. Let us therefore turn to molecular formulas.

We will use the following operators: the singulary operators are the signs for negation '~', existential quantification

---

+The precise definitions, with rules for substitution and evaluation, are given in [11]. In the formal development of [11], I have included predicates in the dictionary. This then allows us to answer questions such as "What relation holds between a and b?'. This type of question provides additional motivation for using the concept of value set.

+When a specific expression is under discussion, we create a name for it by enclosing it in single quotes. The function ω maps expressions into expressions. Thus, the value set of the expression 'Wxb' consists of the symbol 'a', and not what 'a' designates, namely, Scott.

'$(\exists x)$', and universal quantification '(x)'; the binary operators are the signs for conjunction '.', disjunction '$\vee$', implication '$\supset$', equivalence '$\equiv$', and for reasons of convenience we also include a special operator for the negation of implication '-'. This last operator can be read as 'but not'; it is equivalent to the combination '.~'. A sentential formula then has either one or two components depending on whether the major operator is singulary or binary.

It is now easy to construct formulas for which the invariance breaks down. Such a formula is '~Wxb'. We have

$$\omega^*(\text{'}{\sim}Wxb\text{'}) = \omega(\text{'}{\sim}Wxb\text{'}) \cup \{\text{'*'}\}$$

That is, the value set of '~Wxb' on D_o is the set of dictionary items with the exception of 'a', and the value set on D_o^* (D_o with the addition of '*') is this same set plus the symbol '*'. The formula '~Wxb' can be regarded, in one interpretation, as the symbolic question for 'Who did not write Waverley?'. We take such a question to be "unreasonable" and we eliminate these objectionable symbolic questions by insisting that the invariance equation hold for every data base. The formulas for which this is true are said to be definite. In other words, let D be any data base and let D* be the data base constructed from D by the addition of the symbol '*' to the dictionary of D, then: a formula $\underline{s}$ is definite if, for any data base D,

$$\omega(\underline{s}) = \omega^*(\underline{s})$$

Note that by referring to any data base, we make definitude a logical property of formulas, i.e., independent of any particular facts of the world as reflected in the file composition.

Atomic formulas are always definite. For molecular formulas, it can be shown that the definitude of a formula with definite components is closely related to the distribution of free variables in the components. The free variable set of an atomic formula is simply the set of distinct variables occurring in it (without regard for their order); that of a binary formula is the set union of the free variable sets of the components; that of a negation is the same as that of the component; and finally, the free variable set of a quantification is the free variable set of the component less the quantified variable. The number of variables in the free variable set is called the degree of the formula. Thus, a formula is a sentence, i.e., of degree zero, if and only if its free variable set is null.

Let $\bar{\varphi}(\underline{r})$, $\bar{\varphi}(\underline{s})$ be the free variable sets of the sentential formulas $\underline{r}$ and $\underline{s}$, respectively. We have sufficient conditions for definitude given in the following:

Characterization Theorem.+ If $\underline{r}$ and $\underline{s}$ are definite, then:

a. The conjunction $\underline{r}.\underline{s}$ is definite

b. If $\bar{\varphi}(\underline{s}) \subseteq \bar{\varphi}(\underline{r})$, then $\underline{r} - \underline{s}$ is definite

c. If $\bar{\varphi}(\underline{r}) = \bar{\varphi}(\underline{s})$, then $\underline{r} \vee \underline{s}$ is definite

d. If $\bar{\varphi}(\underline{r}) = \bar{\varphi}(\underline{s})$ = the null set, then $\sim\underline{r}$, $\underline{r} \supset \underline{s}$, and $\underline{r} \equiv \underline{s}$ are definite.

We now have the logical apparatus required to proceed with the study of quantification.

III FUNDAMENTAL RESULTS ON QUANTIFICATION

The first question is: Are quantifications of definite formulas definite? The

---

+For proofs, see [11, p. 51f]. An expanded version of the theorem is given in [12]. The sufficient conditions, involving the free variable sets, are "sharp" in the sense that they are also necessary providing certain exceptional cases do not hold.

We use underlined lower case letters singly and combined with operators as metalinguistic names of general formulas. The single quotes are reserved to create names of particular example expressions. See the previous footnote.

symbolic question

$$(\exists x)Wxy \qquad (5)$$

corresponding to 'Which are publications?' provides an example with an existential quantifier. The component of (5), 'Wxy', is atomic (it corresponds to the question 'Who wrote what?') and hence, as we remarked above, is definite. Now the value set of '$(\exists x)Wxy$' is obtained from the value set of 'Wxy', a set of pairs of names, by deleting the first members of the pairs. It can be shown that the addition of '*', although allowing one more value to be considered in the range of the quantification, will nevertheless have no effect on this calculation; in fact we have the general result [11, p. 61]:

Theorem 1. An existential quantification of a definite formula is always definite.

If we think of definite formulas as somehow "natural," then this result is satisfactory in the following sense: A formula with free variables can be considered to define a relation, and its "domains," as given by existential quantification, are the extensions of "natural" properties. Thus, in our example, '$(\exists x)Wxy$' gives (through a λ-formulation) the predicate 'publications'. Furthermore, there are several important concepts in the theory of relations in which existential quantification is a principal ingredient. For example, the relative product of relations R and S is defined through the formula '$(\exists y)Rxy.Syz$'. If the components are definite, then (by the characterization theorem) so is the conjunction and hence (by Theorem 1) the entire formula.+ Another example is the plural relational description (defining the class of things having a given relation to things in a given class, e.g., 'the publications of American logicians').+ A third application is its use as a linguistic device to state that the intersection of two classes is not empty. An example is 'there are smiling villians' (from Russell [16, p. 83f]). Usually, however, the existential quantifier is masked by the indefinite description. This is an argument expression consisting of an indefinite article followed by a noun. There is a special operator, the η-operator, to symbolize an indefinite description.++ Thus, the phrase 'a book' is symbolized as '$(\eta x)(Bx)$'. The sentence 'Carnap wrote a book' then becomes

$$Wc(\eta x)(Bx)$$

The transformation rule for η-expressions+++ then yields the formula

$$(\exists x)Bx.Wcx$$

which states that the class of publications of Carnap and the class of books have a non-null intersection. This conjunctive use of existential quantifiers we regard as fundamental. In other words,

---

+See [18, vol. I, p. 256f]. Note that the relative sum [7, 17] given through '$(y)(Rxy \vee Syz)$' does not seem useful except in a formal way. There is no structure of natural language to which it corresponds. It does not surprise us therefore to find that it is indefinite.

+See [18, vol. I, p. 279f].

++We use the η-operator in the sense of [15, p. 264f] rather than in the sense of Hilbert as described in [7, p. 64-65].

+++The η-operator (as other descriptive operators) is embedded in its scope. To avoid a scope designation, we take the scope to be the smallest primitive predicate in which the argument expression occurs. For this purpose, we regard the identity relation as primitive; but diversity as compound. The rule is: If F is primitive and '$(\eta x)(Gx)$' occurs in the i^{th} argument position of F, then $F...(\eta x)(Gx)...$ is equivalent to $(\exists x)Gx.F...x...$ where 'x' occurs in the i^{th} argument position of F.

given a simple atomic predicate expression, say 'Fx', we can think of this as corresponding to a "natural" property. Then '$(\exists x)Fx$' does not say much, merely that the property occurs (i.e., the extension is not empty). However, given an additional predicate expression 'Gx', then '$(\exists x)Fx.Gx$' communicates non-trivial information. By our theorems, this fundamental use is definite.+

Consider next the universal quantification, $(x)\underline{s}$, of a formula $\underline{s}$, where $\underline{s}$ is a sentential formula with a free variable x (and perhaps other free variables). Suppose that $\underline{s}$ is definite. Then certainly on D* '*' will not be in the value set of s; hence $(x)\underline{s}$ will be false on D* (i.e., have value '0'). If $(x)\underline{s}$ is definite, then by definition $(x)\underline{s}$ is also false on D. Conversely, if $(x)\underline{s}$ is false on D, it is certainly false on D*; hence its value is unchanged. This same argument holds if $\underline{s}$ has other variables besides 'x'; we merely replace 'false' with 'has a null value set'. Let us call a formula a c-formula if, for every data base, it either has value '0' (if it is a sentence), or it has a null value set (if of degree greater than zero). Thus, c-formulas correspond to contradictory formulas; on no data bases are there substitutions which make them true. We have shown the following result:

Theorem 2. A non-vacuous universal quantification of a definite formula is definite if and only if the result is a c-formula.

Now, c-formulas do not correspond to questions or assertions having a great deal of empirical value. It therefore follows from Theorem 2 that the practical use of universal quantification is over indefinite formulas. This is indeed the case. The principal use of universal quantification is over an implication and it turns out that an implication with definite components and at least one free variable is always indefinite providing that an extraordinary situation does not occur in which the antecedent is a true sentence on every data base [11, p. 59].

+Note that existential quantification over a disjunction is of even less interest than quantification of a one-place primitive predicate. This is because existential quantification is distributive over disjunction.

IV RESTRICTED OPERATORS

Before turning to an examination of universal quantification over implication, let us review the notion of restricted quantified variable [15, p. 162f].+ These variables correspond to argument expressions such as 'every F', 'all F's','some F's'. (The last is closely related to the indefinite description.) We indicate the restriction to a class F by a superscript. Thus, the restricted existential quantifier is '$(\exists x)^F$', and the restricted universal quantifier is '$(x)^F$'.The transformation rules are as follows: If we apply the operators to the formula 'Gx', then

$$(x)^F Gx \qquad (6)$$

becomes

$$(x)(Fx \supset Gx) \qquad (7)$$

and

$$(\exists x)^F Gx \qquad (8)$$

becomes $(\exists x)(Fx.Gx)$ (9)

The importance of these operators for applied logic lies in the fact that if the class F is not empty, then the rules of ordinary quantification are carried over for restricted quantifiers. Because of this, it seems that phrases in conversational language that designate restricted universal quantifiers have a secondary meaning component suggesting

+A reviewer has pointed out that a discussion of restricted variables can also be found in Rosser's "Logic For Mathematicians" (p. 140 ff).

that the restrictive class is non-empty. We shall examine this more closely in Sec. VI.

Frequently, the restricted universal quantifier appears in the form of a restricted free variable; i.e., the scope of the quantification is the entire formula. Such a variable we express as 'x^F'. Thus, if

$$\ldots x^F \ldots \qquad (10)$$

stands for 'x^F' in its entire context as represented by the dots, then (10) is transformed into

$$Fx \supset \ldots x \ldots$$

and by the rule of free variables into

$$(x)(Fx \supset \ldots x \ldots) \qquad (11)$$

In a sense, we can regard the restricted free variable---as it appears in an argument expression---to be analogous to the indefinite description. It is appropriate to call it an all description.

V UNIVERSAL QUANTIFICATION OVER IMPLICATION

Using restricted variables and indefinite descriptions, we can symbolize question (2) 'Is every F a G?' as

$$x^F = (\eta y)(Gy) \qquad (12)$$

where 'F' is an abbreviation for '$(\lambda y)(Wey)$' and 'G' replaces 'B'. Thus, by first eliminating the indefinite description, we get

$$(\exists y)(Gy.y = x^F)$$

But this is equivalent to

$$Gx^F$$

Hence, by the rule for transforming 'x^F', (12) becomes

$$(x)(Fx \supset Gx) \qquad (13)$$

Thus we have derived the paraphrase we spoke of earlier.

The first issue to be resolved is whether or not (13) is definite.

Theorem 3. If $\underline{r}$ and $\underline{s}$ are definite, and $\overline{\Phi}(\underline{r}) = \overline{\Phi}(\underline{s}) = \overline{\{x\}}$, then $(x)(\underline{r} \supset \underline{s})$ is definite.

Proof. $(x)(\underline{r} \supset \underline{s})$ is equivalent to

$$\sim(\exists x)(\underline{r} - \underline{s}) \qquad (14)$$

But $\underline{r} - \underline{s}$ is definite by hypothesis and Part b of the Characterization Theorem; $(\exists x)(\underline{r} - \underline{s})$ is definite by Theorem 1; and $\sim(\exists x)(\underline{r} - \underline{s})$ is definite by Part d of the Characterization Theorem. Hence, Theorem 3.

On the basis of Theorem 3, we see that question (2) is definite. There is now the special problem of the appropriate answer to question (2), under an extensional interpretation, if the class F is empty.+ As our symbolization stands, the answer to (13) would be "true." However, if the semantic analysis of 'F' is given by the restricted quantified variable, then part of the meaning of (12) is that F is not empty (the property F does occur). We will be able to clarify this point if we first study the more complicated third example question, 'Who wrote all the papers on inductive logic?'.

Question (3) receives a symbolization using an all-description. This time let 'Fy' be an abbreviation of '$((\exists z)Wzy - By).Iy$'; i.e., 'Fy' says 'y is a paper indexed under inductive logic' Then the symbolic question for (3) is

$$Wxy^F \qquad (15)$$

('x wrote all F'), and by the rule for restricted variables,

$$(y)(Fy \supset Wxy) \qquad (16)$$

+In an intensional interpretation, the question asks whether the meaning of 'G' is part of the meaning of 'F' independent of any world in which the property F does or does not occur.

Thus (16) is an open universal sentence.

Now, as (16) stands, it is indefinite. This is because, on a data base for which 'Fy' has a null value set, '*' will satisfy (16). If these data bases are excluded through the meaning of the question, then we can show that (16) becomes definite. Note that (16) presents a different order of difficulty from (13); (13) is definite, (16) is not. However, we shall propose a symbolization which treats both in a uniform way.

VI EXTENSIONAL QUANTIFIERS

Let us actually make explicit a secondary meaning component for an all-description 'all F's' that states that the property F occurs. Thus we conjoin to both (13) and (16) the condition

$$(\exists z)Fz$$

This meaning seems appropriate for extensional questions. A formal theory of this will now be given.

Let $\underline{r}$ be a sentential formula and let $\varphi(\underline{r})$ be its free variable sequence, i.e., the free variables that occur in $\underline{r}$ in typographical order without repetitions. Then we use the following abbreviations: by '$(\exists\varphi(\underline{r}))$' we mean the existential quantification of every variable in the sequence $\varphi(\underline{r})$. Thus, if $\varphi(\underline{r})$ is the sequence $x_1, x_2, \ldots, x_n$, then

$$(\exists\varphi(\underline{r}))(\ldots) = (\exists x_1)(\exists x_2)\ldots(\exists x_n)(\ldots)$$

Similarly, we use the abbreviation '$(\varphi(\underline{r}))$' for the corresponding universal quantification; i.e.,

$$(\varphi(\underline{r}))(\ldots) = (x_1)(x_2)\ldots(x_n)(\ldots)$$

We now define

Definition. The Extensional Universal Quantifier

$$(A\underline{r})(\underline{s}) = (\exists\varphi(\underline{r}))(\underline{r}).(\varphi(\underline{r}))(\underline{r} \supset \underline{s}) \quad (17)$$

For example, by (17), '(AFy)(Wxy)', is defined to be an abbreviation for '$(\exists y)(Fy).(y)(Fy \supset Wxy)$'.

Note that there is one universal implication that cannot be "extensionally" given by (17). This is an implication of the form

$$(z)(\underline{r} \supset \underline{s}) \quad (18)$$

where z occurs in $\underline{s}$ but not in $\underline{r}$. In [11, p. 102], I have called such implications degenerate. This is because (18) is equivalent to

$$\underline{r} \supset (z)\underline{s} \quad (19)$$

The consequent of (19), if definite with definite component, would be a c-formula according to Theorem 3.

The extensional quantification+ differs from the definition of restricted quantifiers by including an explicit formulation of the existence condition. We shall compare the two notions below, but we first study the definitude of $(A\underline{r})(\underline{s})$.

The case of vacuous quantification is treated by the following result:

Theorem 4. If the free variable set of $\underline{r}$ is null, then $(A\underline{r})(\underline{s})$ is equivalent to the conjunction $\underline{r}.\underline{s}$.

Proof. The definition of $(A\underline{r})(\underline{s})$ becomes $\underline{r}.(\underline{r} \supset \underline{s})$; hence the result.

The next result is a reduction theorem.

Theorem 5. Let X be the free variable sequence of the variables in $\varphi(\underline{r})$ but not in $\varphi(\underline{s})$; then, if there are variables in X, $(A\underline{r})(\underline{s})$ is equivalent to $(A(\exists X)(\underline{r}))(\underline{s})$.

Proof. Let Y be the sequence of free variables common to $\underline{r}$ and $\underline{s}$. Then we can write the definition (17) as

$$(\exists X)(\exists Y)(\underline{r}).(X)(Y)(\underline{r} \supset \underline{s})$$

or equivalently (since the variables of X do not occur in $\underline{s}$),

+In [12], I called this generalized quantification.

$$(\exists Y)((\exists X)(\underline{r})).(Y)((\exists X)(\underline{r}) \supset \underline{s})$$

and, again by definition (17), as

$$(A(\exists X)(\underline{r}))(\underline{s})$$

Hence, Theorem 5.

We can now prove the main result.

Theorem 6. If $\underline{r}$ and $\underline{s}$ are definite, then $(A\underline{r})(\underline{s})$ is definite.

Proof. First, consider the case where the variables in $\varphi(\underline{r})$ are all common to $\underline{s}$, i.e., in the above terminology these are Y-variables. We then have that $(A\underline{r})(\underline{s})$is equivalent to

$$(\exists Y)(\underline{r}).((\exists Y)(\underline{s}).(Y)(\underline{r} \supset \underline{s})) \quad (20)$$

The first component of (20) is definite by Theorem 1. It is the second component of (20) that we must then show to be definite. Let $\underline{t}$ be $(\exists Y)(\underline{s})$. Then the second component of (20) is equivalent to

$$\underline{t}.(Y)(\underline{r}.\underline{t} \supset \underline{s})$$

But this is

$$\underline{t} - (\exists Y)((\underline{r}.\underline{t}) - \underline{s}) \quad (21)$$

and therefore, because $\underline{r}.\underline{t}$ has all the variables of $\underline{s}$, (21) is definite by the Characterization Theorem and Theorem 1.

For the second case, let there be variables in $\varphi(\underline{r})$ that are not in $\varphi(\underline{s})$, i.e., what we called X-variables above. Then by Theorem 5 we see that $(A\underline{r})(\underline{s})$ is equivalent to

$$(A(\exists X)(\underline{r}))(\underline{s})$$

But the variables in $(\exists X)(\underline{r})$ are all Y-variables, hence, since $(\exists X)(\underline{r})$ is definite by Theorem 1, we again get the first case.

This proves Theorem 6.

Thus we see that if the universal quantifiers of our example questions (2) and (3) are interpreted "extensionally," then (3) is definite, as is (2), and both receive a uniform treatment.

VII A COMPARISON OF EXTENSIONAL QUANTIFIERS WITH RESTRICTED QUANTIFIERS

Note that the extensional existential quantifier $(E\underline{r})(\underline{s})$ defined by analogy with (17) as

$$(\exists \varphi(\underline{r}))(\underline{r}).(\exists \varphi(\underline{r}))(\underline{r}.\underline{s})$$

is equivalent to

$$(\exists \varphi(\underline{r}))(\underline{r}.\underline{s}) \quad (22)$$

Therefore $(E\underline{r})(\underline{s})$ coincides with the restricted existential quantifier.

There are two basic relations between existential and universal quantifiers:

$$(x)\underline{s} \text{ if and only if } \sim(\exists x)(\sim\underline{s}) \quad (23)$$

and

$$(x)\underline{s} \text{ implies } (\exists x)\underline{s} \quad (24)$$

For restricted operators, the relation (23) always holds, while (24) holds if the restriction class is not null.+ For the extensional quantifiers, this situation is reversed---(24) generally holds (because we have made explicit the condition that the restriction class is not null), while (23) requires the assumption of a non-null restriction class. This is for the same reason; since the restriction is explicit it also comes within the scope of the negation. More exactly, we have

Theorem 7. If $(\exists \varphi(\underline{r}))(\underline{r})$, then $(A\underline{r})(\underline{s})$ is equivalent to $\sim(E\underline{r})(\sim\underline{s})$.

Proof. Given

$$(A\underline{r})(\underline{s}) \quad (25)$$

we get by definition (17) and the rule for elimination of the universal quantifier (23),

$$(\exists \varphi(\underline{r}))(\underline{r}) - (\exists \varphi(\underline{r}))(\underline{r} - \underline{s})$$

+See [15, pp. 162-163].

and rewriting by using the extensional existential quantifier,

$$(\exists\varphi(\underline{r}))(\underline{r}) - (E\underline{r})(\sim\underline{s}) \qquad (26)$$

But (26) implies

$$\sim(E\underline{r})(\sim\underline{s}) \qquad (27)$$

To go from (27) back to (25) we must introduce the first component of (26) by hypothesis. Hence the result.

This concludes our theory of quantification in question-answering systems. We propose that, if universal quantification is included in the vocabulary of a query language, then it should take the form of an "extensional universal quantifier" as defined in (17). The rules governing its use should be based on Theorems 4, 5, and 6. A computational rule for evaluation is given in the appendix.

APPENDIX

In Theorem 6 we have given a procedure for computing the value set of $(A\underline{r})(\underline{s})$ in terms of the value sets of $\underline{r}$ and $\underline{s}$. Indeed, (21) is in so-called proper form, i.e., each subformula of (21) is definite.+ However, as a computational procedure, it is not the most efficient; in the proof we were guided by logical simplicity. Let us now take a closer look at the computational aspect. The issue centers on the "restrictive" condition $\underline{t}$ in (21), which we have taken to be

$$(\exists Y)(\underline{s}) \qquad (28)$$

Now, if there are free variables in (28), they are variables in $\varphi(\underline{s})$ but not in $\varphi(\underline{r})$. We call such variables 'Z-variables'. Consider $\underline{s}$ to define a relation between Y-values and Z-values. Then a Z-value that satisfies $(A\underline{r})(\underline{s})$ must lie in the converse domain of $\underline{s}$; i.e., it must satisfy (28). But a Z-value that satisfies $(A\underline{r})(\underline{s})$ must also be a relatum of every Y-value in $\underline{s}$ that also satisfies $\underline{r}$. Suppose that Y_o is in the value set of $\underline{r}$, then, rather than using the converse domain of $\underline{s}$ as a restrictive condition, we use the relatum of Y_o with respect to $\underline{s}$. This last class is expressed by substituting Y_o for Y in $\underline{s}$; i.e., the result we denote by

$$[Y \rightarrow Y_o](\underline{s}) \qquad (29)$$

It is therefore possible to replace $\underline{t}$ as it occurs in (21) by (29). Since (29) represents a much smaller class than (28), this new choice of $\underline{t}$ will give a correspondingly smaller cartesian product $\underline{r}.\underline{t}$ in (21).+ This discussion is summarized in the following:

Rule for Computing $\omega((A\underline{r})(\underline{s}))$.

1) Calculate the sequences X, Y, and Z of free variables.
2) If X is not blank,++ then form

$$\underline{r}' = (\exists X)(\underline{r})$$

By Theorem 5, $(A\underline{r})(\underline{s})$ is equivalent to $(A\underline{r}')(\underline{s})$. The desired value set is found by setting $\underline{r}$ equal to $\underline{r}'$, X equal to blank, and proceeding to Step 3.
3) If X is blank, and if Y is also blank, the value set is (by Theorem 4)

$$\omega(\underline{r}.\underline{s})$$

+For a detailed discussion of the algorithms for determining value sets of definite formulas in proper form see [11, pp. 67-80].

+The value set of the conjunction $\underline{r}.\underline{t}$ is the cartesian product of the value sets of the components. See [11, p. 75]

++If no such variables occur, we speak of the result as being "blank," i.e., the calculation yields the machine-readable character blank.

4) If Y is not blank, calculate $\omega(\underline{r})$: if $\omega(\underline{r})$ is null, then the value set is 0 (if Z is blank) or null (if Z is not blank).

5) If $\omega(\underline{r})$ is not null, then let Y_o be the first member of the value set of $\underline{r}$. Set

$$\underline{t} = [Y \rightarrow Y_o](\underline{s})$$

then calculate the value set as

$$\omega(\underline{t} - (\exists Y)((\underline{r}.\underline{t}) - \underline{s}))$$

This concludes the algorithm.

REFERENCES

[1] Åqvist, L. A New Approach to the Logical Theory of Interrogatives, Part I, Analysis. University of Uppsala, Uppsala, Sweden, 1965.

[2] Belnap, N. D., Jr. An Analysis of Questions: Preliminary Report. System Development Corporation, TM-1287/000/00, June 3, 1963.

[3] Bobrow, D. G. A question-answering system for high school algebra word problems. Proc. FJCC, v. 26, pp. 591-614, Spartan Books, Inc., Baltimore, Maryland, 1964.

[4] Carnap, R. Introduction to Symbolic Logic and Its Applications. Dover Publications, Inc., New York, 1958.

[5] -----. The Logical Syntax of Language. (English Edition). Routledge and Kegan Paul, Ltd., London, 1937.

[6] -----. Meaning and Necessity. (Enlarged Edition 1956). University of Chicago Press, Chicago Illinois, 1956.

[7] Feys, R. and F. B. Fitch (ed). Dictionary of Symbols of Mathematical Logic. North-Holland Publishing Co., Amsterdam,1969.

[8] Harrah, D. Communication: A Logical Model. The M.I.T. Press, Cambridge, Massachusetts, 1963.

[9] Jespersen, O. The Philosophy of Grammar. W. W. Norton and Co., Inc., New York, 1965.

[10] Kasher, A. Data-retrieval by computer: A critical survey. (in Kochen, M., (ed), The Growth of Knowledge, pp. 292-324, John Wiley and Sons, Inc., New York, 1967.

[11] Kuhns, J. L. Answering Questions by Computer: A Logical Study. The Rand Corporation, RM-5428-PR, December 1967.

[12] -----. Interrogating a Relational Data File: Remarks on the Admissibility of Input Queries. The Rand Corporation, R-511-PR, November 1970.

-----. Logical aspects of question answering by computer (to be published in Proc. Third International Symposium on Computer and Information Sciences,Miami Beach Florida, December 18-20, 1969, Academic Press; also available as P-4251, The Rand Corporation, 1969).

[13] Levien, R., and M. E. Maron. Relational Data File: A Tool for Mechanized Inference Execution and Data Retrieval. The Rand Corporation, RM-4793-PR, December, 1965.

[14] -----. A computer system for inference execution and data retrieval. Comm. ACM, v. 10, pp. 715-721, 1967.

[15] Reichenbach, H. Elements of Symbolic Logic. Macmillan Co., New York, 1947.

Rosser, J. B. Logic for Mathematicians. McGraw-Hill, New York, 1953.

[16] Russell, B. An Inquiry into Meaning and Truth. Penguin Books, Baltimore, Maryland, 1962.

[17] Tarski, A. On the calculus of relations. J. Symbolic Logic, v. 6, pp. 73-89, 1941.
[18] Whitehead, A. N., and B. Russell. Principia Mathematica (Second Edition), Cambridge University Press, 1950.

THE RELATIONAL DATA FILE AND THE DECISION PROBLEM FOR CLASSES OF PROPER FORMULAS

Robert A. DiPaola
The Rand Corporation, Santa Monica, California

ABSTRACT

The Relational Data File (RDF) of The Rand Corporation is among the most developed of question-answering systems. The "information language" of this system is an applied predicate calculus. The atomic units of information are binary relational sentences. The system has an inference-making capacity.

As part of the actual construction and implementation of the RDF, a theory was developed by J. L. Kuhns to identify those formulas of the predicate calculus which represent the "reasonable" inquiries to put to this system. Accordingly, the classes of definite and proper formulus were defined, and their properties studied. The definite formulas share a semantic property Kuhns judged as necessarily possessed by a reasonable question to be processed by the RDF. The author has previously shown that the decision problem for the class of definite formulas is recursively unsolvable. The proper formulas are definite, and satisfy additional syntactic conditions intended to make them especially suitable for machine processing. The class of proper formulas depends on which logical primitives are employed. Different primitives give rise to different classes of formulas. A formula which can be effectively transformed into a proper equivalent is admissible. Kuhns conjectures that with respect to one particular class of proper formulas, all definite formulas are admissible. In the paper it is shown that the decision problem for several classes of proper formulas is solvable. The following results are established. Theorem 1: The class of proper formulas in prenex form on any complete set of connectives is recursive. Theorem 2: The class of proper formulas on $\neg$, $\vee$, $\exists$ is recursive. Theorem 3: The class of proper formulas on $\neg$, $\supset$, $\exists$ is recursive. Theorem 4: The class of proper formulas on $\neg$, $\supset$, $\vee$, $\exists$ is recursive. Thus, there is a mechanical decision procedure which determines whether an arbitrary formula is a member of the class. It follows that the analogues of Kuhns' conjecture for these classes are false.

KEY WORDS AND PHRASES

data retrieval system, Relational Data File, applied predicate calculus, definite formulas, proper formulas, solvable cases of the decision problem, recursively unsolvable decision problem, question-answering system

I. Introduction

The so-called "information language" of Rand's Relational Data File is an applied predicate calculus [3, 4, 5, 6, 7, 8]. As an integral part of the design and implementation of this storage and

Preceding page blank

retrieval system, a theory was developed by J. L. Kuhns to identify and systematically study those formulas of this calculus that represent the "reasonable" questions to put to a computer implementation of this system, with emphasis placed upon those representations that are supposedly especially suited for machine processing [3, 4]. Accordingly, three classes of formulas were defined--definite, proper, and admissible. The definite formulas are defined semantically and are invariant under the sentential and quantificational transformations of the predicate calculus. They share a semantic condition judged as being necessarily possessed by the symbolic representations of reasonable inquiries. To elaborate, the notion of a data base as defined in [3] amounts to the common notion of a structure with a finite number of relations; or, from the vantage point of a formal language, it is an interpretation of a finite number of predicate and constant symbols. The formulas F with free variables that are definite have the property that the sets of true instances of F in an interpretation I of F and in a special extension I' of I are the same. The formulas without free variables are definite if their truth value is always preserved on passage from an interpretation I to an extension I' of I of the aforesaid special type. A precise definition is given in Sec. 2.

The proper formulas are those definite formulas that satisfy a certain syntactic condition--namely, their principal subformulas must also be proper. Thus, all subformulas must be definite. The rationale behind this definition presumably runs as follows: The definite formulas mirror the reasonable inquiries, while the proper formulas are such that all of their parts, i.e., subformulas, also have this desirable property. The admissible formulas are those that can be transformed into proper equivalents; that is, if F is admissible, there is a proper formula G, such that for each interpretation I, with domain D, F and G are satisfied in D at exactly the same instances. It is conjectured in [3] that every definite formula is admissible. Of course, even for one who accepts the theory as sound, sensible, and adequate it is a priori clear that the admissibility of a formula F is of little or no value unless the transformation φ of F into a proper equivalent G is effective.

The concept, then, of a proper formula involves the notion of a subformula. But determining which consecutive parts of a given formula are subformulas is, of course, a syntactic notion and depends on the identity of sentential and quantificational connectives employed. Thus, the subclasses of proper formulas, in contradistinction to the class of definite formulas, depend on which of the logical connectives are taken as primitives. This paper considers the decision problem for various classes of proper formulas. We point out that an affirmative solution to the decision problem for a particular class Π of proper formulas refutes the version of the aforementioned conjecture relative to the class of definite formulas on any complete set of logical connectives; that is, if Δ is a class of definite formulas on any complete set of connectives, it follows, as in [2], that Δ is not re (recursively enumerable). Hence, if Π is recursive, there is no effective transfor-

mation φ such that for each formula F, $F \in \Delta \Leftrightarrow \varphi(F) \in \Pi$.

II. Definitions

The language $\mathfrak{L}$ that we use is the language of the full, pure first-order predicate calculus without equality, augmented with infinitely many individual constants. A formula in the predicate symbols $P_1^{n_1}, P_2^{n_2}, \ldots, P_t^{n_t}$, where the superscript denotes the rank or degree of the predicate symbol, and in the constants $c_1, c_2, \ldots, c_k$ is any formula F whose only symbols, other than sentential connectives, quantifiers, and individual variables, occur among $P_1^{n_1}, P_2^{n_2}, \ldots, P_t^{n_t}, c_1, \ldots, c_k$. An interpretation of F is a system $I = \langle D; R_1^{n_1}, \ldots, R_t^{n_t}, d_1, \ldots, d_k \rangle$, where D is a nonempty set, each relation $R_i^{n_i}$ is defined on D^{n_i} and assigned to $P_i^{n_i}$, and each d_j is assigned to c_j, $i = 1, \ldots, t$; $d_j \in D$, $j = 1, \ldots, k$. An interpretation I of F is said to be finite if the domain D of I is finite. Developments of the notion of interpretation or structure may be found in [9] or [10].

Definition

If F is a formula with m free variables and $I = \langle D; R_1^{n_1}, R_2^{n_2}, \ldots, R_t^{n_t}; d_1, \ldots, d_k \rangle$ is a finite interpretation of F, T(F, I) is the set of members of D^m that satisfy F, if $m > 0$. If $m = 0$, we call F a sentence, and T(F, I) = t (truth) and T(F, I) = f (falsity) according to whether F is satisfied or not satisfied in I. We say a sentence F is finitely satisfiable if there is a finite interpretation in which it is satisfied; F is finitely valid if it is satisfied in all finite interpretations.

Definition

Let F be a formula and $I = \langle D; R_1^{n_1}, \ldots, R_t^{n_t}; d_1, \ldots, d_k \rangle$ a finite interpretation of F. Let * be an individual not in D. A *-extension I' of I for F is the interpretation $I' = \langle D'; S_1^{n_1}, \ldots, S_t^{n_t}; d_1, \ldots, d_k \rangle$, where $D' = D \cup \{*\}$ and $S_i^{n_i}$ is the extension of $R_i^{n_i}$ from D^{n_i} to D'^{n_i}, such that $S_i^{n_i}$ is false on any member of D'^{n_i} that has * among its components.

Definition

A formula F is said to be definite if, for all finite interpretations I of F, T(F, I) = T(F, I'), where I' is a *-extension of I for F.

Thus, the theorems and refutables of the predicate calculus that are sentences are definite. Clearly, all atomic formulas of $\mathfrak{L}$ are definite. Closure properties of the class of definite formulas are investigated at length in [3].

Remark

We recall again the following theorem [2]:

> If A_k, $k \geq 0$, is the class of definite formulas on $\mathfrak{L}$ with k free variables, then A_k is not recursively enumerable.

Thus, the decision problem for each A_k is recursively unsolvable.

Examples

Consider the following formula F : $\forall x \exists y P(x, y) \supset \exists y \forall x P(x, y)$. Let $D = \{1, 2\}$ and P be interpreted in D by the binary relation R true on the pairs (1, 2) and (2, 1), and false on the pairs (1, 1) and (2, 2). Thus, in this case $I = \langle D; R \rangle$. It is easily seen that $\forall x \exists y P(x, y)$ is

true in I, whereas $\exists y \forall x P(x, y)$ is false in I. By the truth-table for implication, $\forall x \exists y P(x, y) \supset \exists y \forall x P(x, y)$ is consequently false in I.

We form the *-extension $I' = \langle D'; S \rangle$ of I by taking $D' = \{1, 2, *\}$ and S to be the binary relation which agrees with R on $D \times D$ and is false on the pairs $(1, *)$, $(*, 1)$, $(2, *)$, $(*, 2)$, and $(*, *)$. Thus, $\forall x \exists y P(x, y)$ is false in I' since $R(*, *)$, $R(*, 1)$, and $R(*, 2)$ are all false in D'. Similarly, $\exists y \forall x P(x, y)$ is false in I'. Hence F is true in I'. Since F is false in I but true in the *-extension I' of I we see that F is finitely satisfiable, is not finitely valid, and is not definite.

Consider the formula G : $\exists x P(x) \ \& \ \forall x [P(x) \supset Q(x)]$. Let D be any non-empty finite set. Suppose $I = \langle D; R_1, R_2 \rangle$ is an interpretation in which R_1 and R_2 are unary relations serving as the interpretations of P and Q, respectively. If R_1 is false throughout D, then $\exists x P(x)$ and hence G are false in I. Thus, G is not finitely valid. But, if, for example, R_1 holds for but a single member of D and R_2 is universally true in D, the G holds in I. Thus, G is finitely satisfiable.

Suppose that $I' = \langle D'; S_1, S_2 \rangle$ is a *-extension of I. Assume that G is false in I. If $\exists x P(x)$ is false in I, then since $S_1(*)$ is false, $\exists x P(x)$ and hence G are false in I'. If $\exists x P(x)$ is true in I but $\forall x (P(x) \supset Q(x))$ is false in I, then there is an element $d \in D$ such that S_1 is true on d and S_2 is false on d. Hence $S_1(d)$ is true and $S_2(d)$ is false in D'. Thus, $\forall x (P(x) \supset Q(x))$ is false in I'. Therefore, G is false in I'. Assume, now, that G is true in I. Then $\exists x P(x)$ is true in I and hence true in I'. By assumptions, $\forall x (P(x) \supset Q(x))$ is true in I. S_1 and S_2 are both false on *. So, $\forall x (P(x) \supset Q(x))$ is true in I'. Therefore G is true in I'. Since I is arbitrary, it follows that G is definite.

Definition

Let $u_1, u_2, \ldots, u_k$ be the unary connectives and $b_1, b_2, \ldots, b_\ell$ the binary connectives of the language $\mathfrak{L}$. It is assumed that at least one of $\exists$ (the existential quantifier) and $\forall$ (the universal quantifier) is among $u_1, u_2, \ldots, u_k$. We inductively define the property of being a subformula of a formula A of $\mathfrak{L}$.

(1) A is a subformula of A.

(2) If A is $u_i(B)$ and u_i is a propositional connective, then each subformula of B is a subformula of A, $i = 1, 2, \ldots, k$.

(3) If A is $u_i x(B)$, u_i is either $\exists$ or $\forall$, and x is a variable that occurs free in B, then each subformula of B is a subformula of A, $i = 1, 2, \ldots, k$.

(4) If A is $b_j(B, C)$, then each subformula of B is a subformula of A and each subformula of C is a subformula of A, $j = 1, 2, \ldots, \ell$.

(5) A formula B is a subformula of A only as prescribed by (1) through (4) above.

Definition

A formula A of $\mathfrak{L}$ is proper iff each subformula of A is definite. In the following, we are mainly interested in proper formulas on the more familiar sets of connectives. If the primitive connectives

are basically those of [3, 4]--namely, $\neg$, $\vee$, &, $\supset$, $\not\supset$ (but not), and $\exists$--the above definition of subformula reads as follows:

(1) A is a subformula of A.

(2) If A is $\neg$ B, then each subformula of B is a subformula of A.

(3) If A is BVC, B&C, B$\supset$C, B$\not\supset$C, then each subformula of B and each subformula of C is a subformula of A.

(4) If A is $\exists$xB, then each subformula of B is a subformula of A.

(5) A formula B is a subformula of A only as prescribed by (1) through (4) above.

Notation

To facilitate the frequent use of certain terms, we shall write "fv," "fs," "nfv," "nfs," "re," for "finitely valid," "finitely satisfiable," "not finitely valid," "not finitely satisfiable," and "recursively enumerable," respectively. Also, we write "$F_1 \overset{I}{\Leftrightarrow} F_2$" to mean that, for each finite interpretation I, the formulas F_1 and F_2 hold at exactly the same instances of the domain of I. If, as is usually the case, $\supset$ is among our primitive connectives, we write "A $\equiv$ B" for "(A$\supset$B)&(B$\supset$A)."

Definition

We define the height h of a formula F.

(1) If F is atomic, h(F) = 0.

(2) If u is a unary primitive connective of the language $\mathfrak{L}$ and F is u(A), then h(F) = h(A) + 1.

(3) If b is a binary primitive connective of $\mathfrak{L}$, and F is b(A, B), then h(F) = max(h(A), h(B)) + 1.

III. Results

Unlike the class of definite formulas, the subclasses of proper formulas depend on which of the logical connectives are taken as primitives. Thus, classes Π_1 and Π_2 of proper formulas employing different but, in the usual sense, equivalent sets of connectives may have different properties. This dependence is, however, not absolute, as the following result demonstrates.

THEOREM 1. The class Π_p of proper formulas in prenex form defined on any complete set of connectives is recursive.

Proof. Consider a formula $F \in \Pi_p$. F is of the form $Qx_1Qx_2 \cdots Qx_mM(x_1, x_2, \cdots, x_m, y_1, y_2, \cdots, y_n)$, where M is quantifier-free and each Q is $\exists$ or $\forall$. Now, it is easy to see that it is decidable whether a quantifier-free formula is fv or nfs. Consequently, by using the various results governing the closure properties of the class of definite formulas given in [3] and [4], p. 11, it is decidable whether M is proper. If M is not proper, F is not proper. So, assume that M is proper. The existential quantification of a proper formula is proper [3]; therefore, if each Q is $\exists$, then F is proper. Suppose that i_o is the greatest integer not exceeding m, such that Qx_{i_o} is $\forall x_{i_o}$; then $\forall x_{i_o} \exists x_{i_o+1} \cdots \exists x_mM$ is proper iff the sentence $\exists y_1\exists y_2 \cdots \exists y_n\exists x_1\exists x_2 \cdots \forall x_{i_o} \exists x_{i_o+1} \cdots \exists x_mM$ is nfs [3], p. 62. Thus, $\forall x_{i_o} \exists x_{i_o+1} \cdots \exists x_mM$ is proper iff the sentence $\forall y_1\forall y_2 \cdots \forall y_n\forall x_1\forall x_2 \cdots \exists x_{i_o} \forall x_{i_o+1} \cdots \forall x_m\neg M$ is fv. But it is decidable

whether such sentences are fv [1], pp. 67-70, 72-74. Hence, it is decidable whether $\exists y_1 \exists y_2 \cdots \exists y_n \exists x_1 \exists x_2 \cdots \forall x_{i_o} \exists x_{i_o+1} \cdots \exists x_m M$ is fs. If the latter sentence is fs, then $\forall x_{i_o} \exists x_{i_o+1} \cdots \exists x_m M$ is not proper, and hence F is not proper. If it is nfs, then $\forall x_{i_o} \exists x_{i_o+1} \cdots \exists x_m M$ is proper. However, if $\exists y_1 \exists y_2 \cdots \exists y_n \exists x_1 \exists x_2 \cdots \forall x_{i_o} \exists x_{i_o+1} \cdots \exists x_m M$ is nfs, it follows that $Qx_j Qx_{j+1} \cdots \forall x_{i_o} \exists x_{i_o+1} \cdots \exists x_m M$ is nfs, $j = i_o-1, i_o-2, \ldots, 1$, and hence is proper. Consequently, if $\exists y_1 \exists y_2 \cdots \exists y_n \exists x_1 \exists x_2 \cdots \forall x_{i_o} \exists x_{i_o+1} \cdots \exists x_m M$ is nfs, then F is proper. Q.E.D.

For the next result, we denote by "Π" the class of proper formulas on $\neg$, $\vee$, $\exists$.

LEMMA 2. If $F \in \Pi$, then it is decidable whether F is fv or F is fs.

Proof. Suppose F contains no occurrence of negation signs. Then plainly F is fs. Also, the closure of a prenex form of F is of the form $\forall x_1 \forall x_2 \cdots \forall x_m \exists y_1 \exists y_2 \cdots \exists y_n M$, where $x_1, x_2, \ldots, x_m$ are the free variables of F. It is decidable whether sentences of this form are fv [1], pp. 70-71. Consequently, it is decidable whether F is fv.

We henceforth assume that F contains occurrences of negation signs. Without loss of generality, we assume that no double negations occur in F. We employ induction on the height h of F. By our assumption, since F is proper, the induction starts with h = 2. For h = 2, F is of the form $\neg \exists x P(x)$, where $P(x)$ is atomic. It is, of course, decidable whether such a formula is fv or fs. Assume that the lemma holds for all formulas F in Π of height $h \leq k$. Consider a formula F of Π of height $h = k + 1$. Suppose F contains a subformula of the form $\neg \exists u A \vee B$, where B has free variables. Since this subformula is proper, $\neg \exists u A$ must be nfs and hence $\neg \exists u A \vee B \overset{I}{\Leftrightarrow} B$. In fact, suppose F' is the formula obtained from F by replacing this particular occurrence of $\neg \exists u A \vee B$ by B. It is easy to see that F' is proper. Moreover, $F' \overset{I}{\Leftrightarrow} F$. F' is of height $h \leq k + 1$; hence, by the induction hypothesis, it is decidable whether F' is fv or fs. Consequently, it is decidable whether F is fv or fs.

We now assume that F contains no subformula of the form $\neg \exists u A \vee B$ (or $B \vee \neg \exists u A$), where B has free variables. Consider all occurrences of subformulas of F that have a single occurrence of a negation sign and that have negation as the senior connective. Such subformulas are sentences of the form $\neg \exists x_1 \exists x_2 \cdots \exists x_m M$, if we ignore the trivial case of the negation of proposition letters. If F is of such a form, then it is decidable whether F is fv or fs without use of the induction hypothesis [1], pp. 62-63. Suppose, therefore, that F is not of this form and such that A is a subformula of F of this type--that is, a sentence $\neg \exists x_1 \exists x_2 \cdots \exists x_m M$. We assert that A lies within the scope of no other negation sign. Suppose it did. Then there is a subformula $\neg \exists x B$ of F, such that A is a subformula of B. Since x occurs free in B, there is a subformula of B of the form $C \vee A$ or $A \vee C$, where C has free variables. This contradicts our assumption that F contains no subformulas of this form; our assertion is therefore justified.

Moreover, for the same reason, A lies within the scope of no other existential quantifier. Let $A_1, A_2, \ldots, A_k$ all be sequential occurrences of such sentences that occur as subformulas of F. Consider the left-most occurrence A_1 and next-to-left-most occurrence A_2 of such sentences. Since F is a formula on $\neg$, $\vee$, $\exists$, there are proper formulas B_1 and B_2 such that $B_1 \vee A_1 \vee B_2 \vee A_2$ is a subformula of F. The binary connective $\vee$ may occur in B_1 and B_2, but no negation signs occur in either. Hence, each occurrence of an existential quantifier in either of B_1 or B_2 is unnegated. The same considerations apply to A_2 and A_3, A_3 and A_4, ..., A_{k-1} and A_k. Thus, F is a formula of the form $B_1 \vee A_1 \vee B_2 \vee A_2 \vee \ldots \vee A_{k-1} \vee B_k \vee A_k \vee B_{k+1}$, using the associativity of $\vee$. Negation signs occur only in $A_1, A_2, \ldots, A_k$, and then only as senior connectives. (Of course, various of the formulas $B_1, B_2, \ldots, B_{k+1}$ may be empty, or F may be of the $B_1 \vee A_1$ for a nonempty B_1; however, in such cases, the situation is yet simpler and the same analysis applies.) F thus has a prenex form (on $\neg$, $\vee$, $\exists$, $\forall$) in which each universal quantifier precedes each existential quantifier. The finite validity of such formulas is decidable [1], pp. 70-71. Similarly, F has a prenex form in which each existential quantifier precedes each universal quantifier. The finite satisfiability of such formulas is decidable. Consequently, it is decidable whether F is fs or fv. Q.E.D.

We define the propositional formula P(h): If F is a formula (on $\neg$, $\vee$, $\exists$) of height h and each subformula of F of height $k < h$ is proper, then it can be effectively determined whether F is proper.

LEMMA 3. P(h) is true for each h.

Proof. P(0) certainly holds, since atoms are proper and h = 0 implies that F is atomic. Assume P(h) for $h \leq k$ and consider P(k + 1). Let F be a formula of height k + 1, such that each subformula A of F of height $h \leq k$ is proper.

Case 1. F is $\neg$ A. Then if A is a sentence, F is proper. Otherwise, F is not proper.

Case 2. F is $\exists x A$, where x occurs free in A. Then F is proper.

Case 3. F is $A \vee B$. If the A and B have the same free variables, then F is proper [3]. By Lemma 2, it can be decided whether A is fs or B is fs. If both A and B are nfs, then F is proper. If one of A and B, say A, is nfs and each free variable of A is a free variable of B, then F is proper [3]. On the other hand, if A and B do not have the same free variables, and (1) both A and B are fs, or (2) just one, say B, is fs but A has free variables that do not occur in B, then F is not proper. Q.E.D.

We therefore have

THEOREM 4. The class of proper formulas of $\vee$, $\neg$, $\exists$ is recursive.

Proof. Consider a given formula F of height h. By 3, P(k) holds for each k. The atoms of F are proper. Thus, it can be determined whether the subformulas of height 1, 2, ..., h of F are proper. Hence, it can be determined whether F is proper. Q.E.D.

We now take up the decision problem for the class of proper formulas on $\neg$, $\supset$, $\exists$.

We define a transformation φ of the formulas on $\neg$, $\supset$, $\exists$ into the formulas on $\neg$, $\vee$, $\exists$.

$$\varphi(A) = A \text{ if } A \text{ is atomic},$$
$$\varphi(\neg A) = \neg \varphi(A),$$
$$\varphi(\exists x A) = \exists x \varphi(A),$$
$$\varphi(A \supset B) = \varphi(\neg A) \vee \varphi(B).$$

LEMMA 5. A is proper on $\neg$, $\supset$, $\exists$ iff $\varphi(A)$ is proper on $\neg$, $\vee$, $\exists$.

Proof. The proof proceeds by induction on the height h of A. If h = 0, then A is atomic, and $\varphi(A) = A$; hence, the assertion holds. Suppose the statement is true for all formulas B of height $h \leq k$. Consider a formula A of height $h = k + 1$.

Case 1. A is $\neg B$. Then $\varphi(A)$ is $\neg \varphi(B)$. Assume A is proper; then B is proper. Then, by the induction hypothesis, $\varphi(B)$ is proper. Since $\neg B$ is proper, A is a sentence. Hence, $\varphi(B)$ is a sentence and $\varphi(A)$ is proper. Likewise, if $\varphi(A) = \neg \varphi(B)$ is proper, $\varphi(B)$ is proper and a sentence. By definition of φ, B is a sentence; by the induction hypothesis, B is proper. Hence, A is proper.

Case 2. A is $\exists x B$. Then $\varphi(A) = \exists x \varphi(B)$. A and $\varphi(A)$ are proper iff B and $\varphi(B)$ are proper. By the induction hypothesis, B is proper iff $\varphi(B)$ is proper. Hence, A is proper iff $\varphi(A)$ is proper.

Case 3. A is $B \supset C$. Then $\varphi(A)$ is $\neg \varphi(B) \vee \varphi(C)$.

Subcase 1. B and C are both sentences. Then $\varphi(B)$ and $\varphi(C)$ are sentences. A is proper iff both B and C are proper. By the induction hypothesis, B is proper and C is proper iff $\varphi(B)$ is proper and $\varphi(C)$ is proper, respectively. Since B is a sentence, B is proper iff $\neg \varphi(B)$ is proper. But $\neg \varphi(B) \vee \varphi(C)$ is proper iff $\neg \varphi(B)$ and $\varphi(C)$ are proper [3, 4]. Therefore, A is proper iff $\neg \varphi(B) \vee \varphi(C)$ is proper.

Subcase 2. At least one of B, C is not a sentence. In this subcase, A is proper iff B is a proper fv sentence and C is proper. Thus if A is proper, C is not a sentence. Hence, $\varphi(C)$ is not a sentence and, by the induction hypothesis, $\varphi(C)$ is proper. Also, if B is a proper, fv sentence, then $\neg \varphi(B)$ is a nfs proper sentence. Hence, $\neg \varphi(B) \vee \varphi(C)$ is proper [4], p. 11. Similarly, since one of B, C is not a sentence, if $\neg \varphi(B) \vee \varphi(C)$ is proper, then $\neg \varphi(B)$ is a proper sentence [3], p. 51, $\varphi(C)$ has free variables, and hence at least $\neg \varphi(B)$ is nfs [4], p. 11. Hence, B is, by the induction hypothesis, a proper sentence and C is a proper formula with free variables. Since $\vdash F \equiv \varphi(F)$ in the predicate calculus for each formula F, B is fv. Hence, A is proper [4], p. 11. We conclude that A is proper iff $\neg \varphi(B) \vee \varphi(C)$ is proper. Q.E.D.

We thus obtain

THEOREM 6. The class of proper formulas on $\neg$, $\supset$, $\exists$ is recursive.

Proof. Indeed, an arithmetization of our formalism, together with Lemma 5, shows that the decision problem for the class of proper formulas on $\neg$, $\supset$, $\exists$ is 1-1 reducible to the decision problem for the class of proper formulas on $\neg$, $\vee$, $\exists$. By Theorem 4, the latter class is recursive. Hence, the class of proper formulas on $\neg$, $\supset$, $\exists$ is recursive. Q.E.D.

The above statement is easily extended to give

THEOREM 7. The class of proper formulas on $\neg$, $\vee$, $\supset$, $\exists$ is recursive.

Proof. We first extend the definition of φ by inclusion of the clause,

$$\varphi(A \vee B) = \varphi(A) \vee \varphi(B).$$

One more case arises in addition to the three considered in 5:

Case 4. A is B$\vee$C. Then $\varphi(A) = \varphi(B) \vee \varphi(C)$. By the induction hypothesis, B is proper iff $\varphi(B)$ is proper and C is proper iff $\varphi(C)$ is proper [4], p. 11.

Subcase 1. B and C have the same free variables. Then $\varphi(B)$ and $\varphi(C)$ have the same free variables; hence, A = B$\vee$C is proper iff $\varphi(B) \vee \varphi(C)$ is proper.

Subcase 2. B and C do not have the same free variables. Since $\vdash C \equiv \varphi(C)$ and $\vdash B \equiv \varphi(B)$ in the predicate calculus, then (1) B and C are nfs iff $\varphi(B)$ and $\varphi(C)$ are nfs and (2) just one, say B, of B, C is nfs and each free variable of B is a free variable of C iff just one, say $\varphi(B)$, of $\varphi(B)$, $\varphi(C)$ is nfs and each free variable of $\varphi(B)$ is a free variable of $\varphi(C)$. So, again, B$\vee$C is proper iff $\varphi(B) \vee \varphi(C)$ is proper. Q.E.D.

IV. Remarks

Reducibility

We denote by "Δ_p", "Δ_1", "Δ_2", "Δ_3" the classes of definite formulas in prenex form, on $\neg$, $\vee$, $\exists$, on $\neg$, $\supset$, $\exists$, and on $\neg$, $\vee$, $\supset$, $\exists$, respectively; by "Π_p", "Π_1", "Π_2", "Π_3", we denote the classes of definite formulas in prenex form on $\neg$, $\vee$, $\exists$, on $\neg$, $\supset$, $\exists$, and on $\neg$, $\vee$, $\supset$, $\exists$, respectively. In contrast to Theorem 1, Theorem 4, Theorem 6, and Theorem 7, the decision problem for each of Δ_p, Δ_1, Δ_2, and Δ_3 is recursively unsolvable. In fact, none of these classes of definite formulas is recursively enumerable but in fact each is a set of the highest degree of unsolvability for sets expressible by the Π_1^0 predicates of the Kleene hierarchy [2], [10]. It follows that the analogue of the conjecture of Kuhns described in the Introduction is false for all combinations of Δ's and Π's; that is, if we momentarily denote by "Δ" any of Δ_p, Δ_1, Δ_2, Δ_3 and by "Π" any of Π_p, Π_1, Π_2, Π_3, then there is no effective transformation φ such that

$$F \in \Delta \Leftrightarrow \varphi(F) \in \Pi \text{ and } F \overset{I}{\Leftrightarrow} \varphi(F).$$

In fact, none of Δ_p, Δ_1, Δ_2, Δ_3 is many-one reducible, or even Turing reducible to any of Π_p, Π_1, Π_2, Π_3. For, if one of Δ_p, Δ_1, Δ_2, Δ_3 were Turing reducible to one of Π_p, Π_1, Π_2, Π_3, then that one of Δ_p, Δ_1, Δ_2, Δ_3 would be recursive, which is not the case. The final general statement along these lines one can make is that none of Δ_p, Δ_1, Δ_2, Δ_3 is reducible to any of Π_p, Π_1, Π_2, Π_3 by a recursively enumberable function, where we understand a function f to be re if the two-place relation $f(x) = y$ is re. Thus, there is no re function f, such that $F \in \Delta \Leftrightarrow f(F) \in \Pi$.

A Concluding Comment

None of the results, Theorem 1, Theorem 4, Theorem 6, Theorem 7, is *a priori* obvious. However, if one inspects the solvable cases of the decision problem [1] on which the proofs of these results are based, one sees that the algorithms that solve the decision problems of Π_p, Π_1, Π_2, Π_3, respectively,

are reasonably simple and "programmable." There is no factor intrinsic to the algorithms that would render implementation infeasible. Computational difficulties would stem rather from testing a formula for finite validity or finite satisfiability when the data base is large.

REFERENCES

(1) ACKERMANN, W. "Solvable Cases of the Decision Problem," in L. E. J. Brouwer, E. W. Beth, A. Heyting (eds.), Studies in Logic and the Foundations of Mathematics, North-Holland Publishing Co., Amsterdam, 1954.

(2) DIPAOLA, R. A. "The Recursive Unsolvability of the Decision Problem for the Class of Definite Formulas," JACM, Vol. 16, No. 2, April 1969, pp. 324-327; also, The Rand Corporation, RM-5639-PR, April 1968.

(3) KUHNS, J. L. Answering Questions by Computer: A Logical Study, The Rand Corporation, RM-5428-PR, December 1967.

(4) KUHNS, J. L. Interrogating a Relational Data File: Remarks on the Admissibility of Input Queries, The Rand Corporation, R-511-PR, November 1970.

(5) LEVIEN, R. E. Relational Data File: Experience with a System for Propositional Data Storage and Inference Execution, The Rand Corporation, RM-5947-PR, April 1969.

(6) LEVIEN, R. E. "Relational Data File II: Implementation," in G. Schector (ed.), Information Retrieval: A Critical View, Thompson Book Co., Washington, D. C., 1966.

(7) LEVIEN, R. E., and M. E. MARON "A Computer System for Inference Execution and Data Retrieval," Comm. ACM, Vol. 10, No. 11, November 1967, pp. 715-721.

(8) MARON, M. E. "Relational Data File I: Design Philosophy," in G. Schector (ed.), Information Retrieval: A Critical View, Thompson Book Co., Washington, D. C., 1966.

(9) MENDELSON, E. Introduction to Mathematical Logic, D. Van Nostrand Co., New York, 1964.

(10) SHOENFIELD, J. R. Mathematical Logic, Addison Wesley, Reading, Mass., 1967.

MANAGING SEMANTIC DATA IN AN ASSOCIATIVE NET

Stanley Y. W. Su
University of Florida, Gainesville, Florida

ABSTRACT

This paper describes the design and implementation of a general associative net structure to be used in an interactive information system, and presents a scheme designed to manage large quantities of semantic data stored in a data base on disc. The associative-net-structured data base is functionally divided into two pools: the hierarchy pool and the linguistic pool. The network of items in the hierarchy pool represents the descriptive information about documents and the network of items in the linguistic pool represents the syntactic and semantic properties of the items in the hierarchy pool. Two search functions and a general search algorithm are presented in this paper. In the implementation, the data base is a regional data set on disc. Items and their associated labeled links are stored on disc tracks. The system establishes a directory to keep track of the items which have associated information stored on more than one track. The use of the directory eliminates unnecessary disc accesses and allows the system to move a proper track into core storage for data processing.

KEYWORDS AND PHRASES

Information storage and retrieval, semantic data structures, hierarchy, attribute-value pairs, disc, data management, associative net, dependency structure, semantics.

1. Introduction

The control and handling of information flow has become one of the most pressing needs in our technologically oriented society. In the past several years research on information storage and retrieval has been very intensive in effort and broad in scope. Existing research can broadly be divided into two major classes based on the different languages used in the systems. The first class can be represented by operational systems such as the MEDLARS(2), the NASA file(4) the SMART system(9) and the Library of Congress System(18). These systems use index languages containing index terms for describing and representing document content. The other class of research concerns the use of natural English as the communication language and the implementation of systems which process natural language materials. Thorough syntactic and semantic analysis of language material is the main emphasis of the second class of research. Surveys in this area have been reported by Simmons(23), Kasher(11) and Simmons(22).

In order for an ISR system to retrieve relevant information in response to a query, it must be able to decipher the semantic content of the query as well as the information stored. One characteristic of the current natural language processing systems is the representation of semantic content by some type of data structures. Answering questions about the stored information or entering information into a data base involve transferring queries or English statements into some formal statements, such as procedural statements in Woods(29) and Kellogg(13) mathematical equations in Bobrow(3) and Charniak (5), flowcharts of computer programs in Kochen(14) or predicate calculus statements in Coles(7), Green and Raphael(8). Some of the data structures used are the ring structure in Lockemann and Thompson(15), variations of triangle structures in Longyear(16), structures under the general conception of nested attribute-value lists in Colby(6), Abelson(1), Simmons(24), Quillian(19) and cylinder structure in Weston and Taylor(28). Systems using these data structures have been tested on small data bases and have been successful in handling information in a restricted domain of discourse. However, whether the data structures used are general enough to handle different types of information remains to be shown. I believe that a data structure more general than the existing ones should be used in a system which aims at handling a great variety of information.

In most existing systems dealing with natural language, data are stored in core storage. The practical problems of how to physically arrange the semantic data and how to get the data out of the storage at reasonable cost and time are generally ignored. In a recent survey, Simmons(22) pointed out the need of a method for managing semantic data in a large data base on auxiliary storage.

This paper presents a general associative net structure to be used in interactive sequential feedback information retrieval system (INSEFIRE) currently under development, and describes a scheme used to manage large quantities of semantic data on disc. The second section of this paper describes the associative net structure and presents two search functions and a search algorithm. The third section describes the implementation of the associative net structure and the scheme used to manage semantic data on disc.

2. Semantic Data Structure

The semantic data structure to be described is designed for storing and processing different types of data such as payroll, personnel records or documents. In this paper, the storage and processing of scientific documents is chosen as an example. The design of the data structure is in-line with the characteristics of an interactive sequential feedback information retrieval system(INSEFIRE) currently under development. It facilitates INSEFIRE to perform the following tasks:

(a) To automatically process the descriptive information of documents which may be presented to the system in the form of a hierarchical collection of items.

(b) To answer the user's specific questions posed in English regarding the descriptive information and the documents stored.

(c) To allow the user to search up and down a hierarchical structure of items which identify different documents.

(d) To decode the information stored in the data structure and to generate specific questions to the user whose answers (feedback) will help the system optimize its search.

2.1 Definitions

In building a data based information retrieval system, the first problem generally concerns the choice and organization of the "basic data units" and "relations" in the data base. Some basic data units and relations are defined below.

(a) Item: An item is the basic data unit which represents the concept of a word, a phrase or any linguistic unit used to describe a document. There is a many-to-one mapping from linguistic units to items.

(b) Record: A record is a hierarchical collection of items. At the bottom of the hierarchy is a collection of items in attribute-value form. Each attribute can have more than one value. The following structure is the external representation of a record.

```
1 Record of a document
  2 record number: 2
  2 author: John Smith
  2 title: On Semantic Data Structures
  2 date of publication
    3 day: 10
    3 month: July
    3 year: 1966
  2 subject: information systems
```

(c) Document: A document is any published material which consists of descriptive information and a text. The descriptive information forms a record which consists of such information as the title, the table of contents and the bibliographic data of a document.

(d) File: A file is a set of records having the same record name. The records may have different hierarchical structures with different attribute-value pairs. For example, the following record belongs to the same file as the previous one.

```
1 Record of a document
  2 record number: 17
  2 author: Robert Jones
  2 title: Computer Software
  2 date of publication: July 10, 1968
  2 publisher: Wiley
```

(e) Relation: Let R be a set of relations which are arbitrary symbols marking the hierarchical relation, the attribute-value relation and the syntactic or semantic relations between items. For example, h specifies the hierarchical relation, a the attribute-value relation and ag the relation 'agent' between a noun and a verb. Let R' be a set containing the converse relation of R. If an item x is related to an item y through h, then y is related to x through h'.

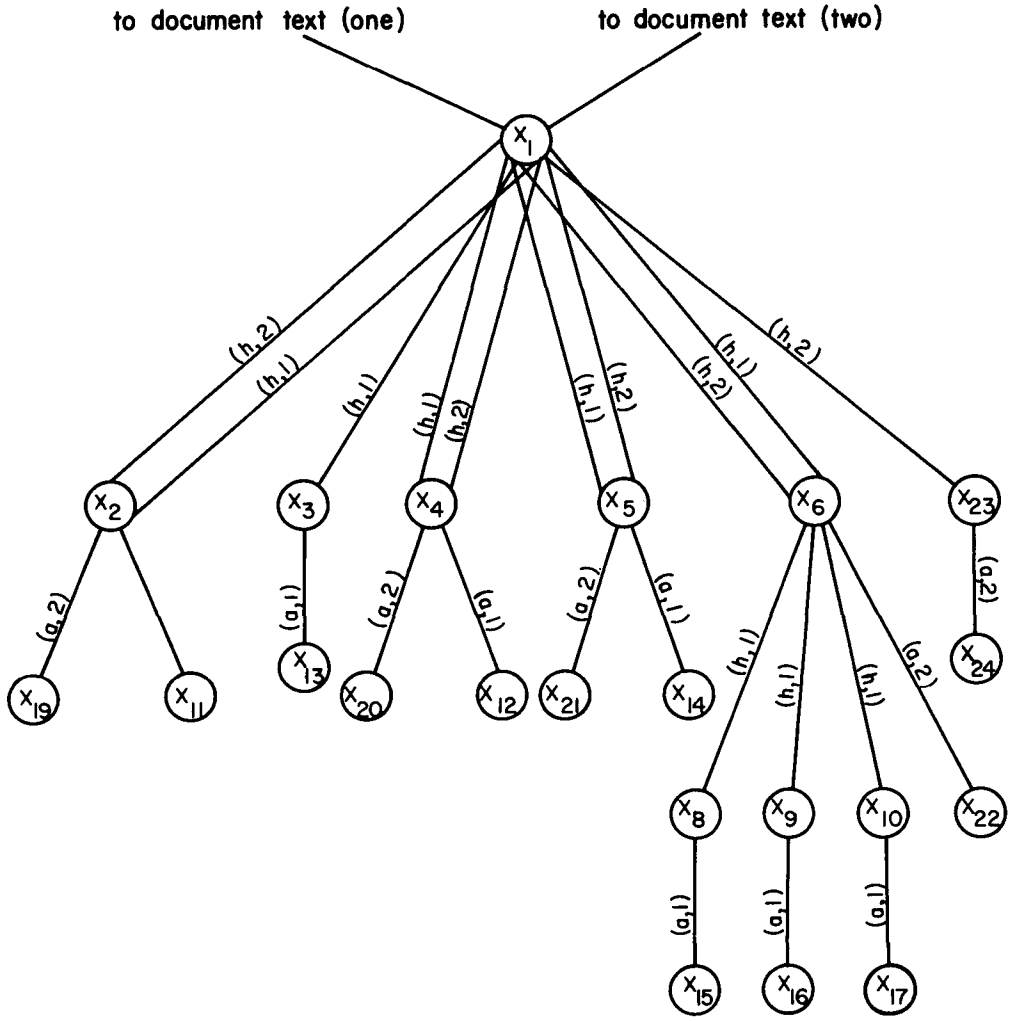
(f) Tag: Let T be a set of positive integers which are used as tags to distinguish the data of different records or the tokens of words occurring in different items.

(g) Label: A label is a member of a subset of the Cartesian product R x T containing a set of ordered pairs of the form (r,t) where $r \in R$ is a relation and $t \in T$ is a tag. A label marks the relation between two items and specifies a particular record in which the relation exists between the items.

2.2 An Associative Net Structure

The associative net structure consists of a complex network of items that can enter into a number of different relations. The items in the structure are associated with one another through the use of labelled links which specify the relationship among the items. The data base using the associative net structure is functionally divided into two data pools: the hierarchy pool and the linguistic pool. The items in the hierarchy pool are those shown in the input records. They are connected to one another with labels marking their hierarchical relations and their attribute-value relations. The items in the linguistic pool are the constituents of the items in the hierarchy pool. They are connected through explicit labels which mark their semantic and syntactic properties of the items in the hierarchy pool(i.e. the items used to represent the descriptive information of documents).

(a) The Hierarchy Pool: In the hierarchy pool, items in the input records are arranged with labels which mark their hierarchical relations and their attribute-value relations. For example, the records whose external representations are shown in our previous examples can be represented graphically in Figure 1. Figure 1 shows that all items in a record are identified by the same tag on the labels. The relation names on the labels specify the hierarchical relation h and the attribute-value relation a among the items. The coded labels can be used



x_1	=	record of document	x_{13}	=	information systems
x_2	=	record number	x_{14}	=	On Semantic Data Structures
x_3	=	subject	x_{15}	=	10
x_4	=	author	x_{16}	=	July
x_5	=	title	x_{17}	=	1966
x_6	=	date of publication	x_{19}	=	17
x_8	=	day	x_{20}	=	Robert Jones
x_9	=	month	x_{21}	=	Computer Software
x_{10}	=	year	x_{22}	=	July 10, 1968
x_{11}	=	2	x_{23}	=	publisher
x_{12}	=	John Smith	x_{24}	=	Wiley

Figure 1. Two Records in the Hierarchy Pool

for decoding the records in the proper format. The x's are used to identify different items in the pool. Their subscripts do not have any special meaning other than to distinguish one x from the other. The node x_1 represents the record name which is the head of all records in a file. It is used as the identifier of the file. In the associative net structure, all labeled links are bidirectional. If x_1 is linked to x_3 through the label (h,1), then x_3 is linked to x_1 through the label $(h^{\vee},1)$ in which the converse relation $h^{\vee}$ is used. The converse relations allow the system to find the governors of items in the hierarchical structure. In actual implementation, an item is represented in storage by a block of computer words containing labels and pointers to other items. The items are identified by the addresses of the blocks.

The following notation shows the relationship between two items.

$r(x,y;t)$ or $r^{(1)}(x,y;t)$, if x is directly linked to y through the label (r,t) where $r \in R \cup R^{\vee}$ represents a relation and $t \in T$ represents a tag.

$r^{(n)}(x,y;t)$ if $r(x,x_1;t)$, $r(x_1,x_2;t)$,.... and $r(x_{n-1},y;t)$.

If n, r or t is replaced by a *, the notation would indicate that x is linked to y through any number of levels, any relation or any tag.

2.3 Search Functions

In the following paragraphs I shall define two functions to be used by the system to retrieve items from the associative net structure. These functions allow the system user to scan the surrounding items of any identified item in the hierarchical structure and thus allows him to decide whether a document is relevant on the basis of the descriptive information he obtains.

(a) Let $IL(x,r,t,n) = \{y \mid r^{(n)}(x,y;t)\}$ be a function with four parameters; x an item, r a relation, t a tag and n a specified level. The function returns as its value a set of items traceable from a given item x through a specified label (r,t) within n levels. The trace can be up or down the hierarchical structure depending on whether r is in R or in $R^{\vee}$. For example, according to Figure 1, we will have the following:

$$IL(x_1,h,2,1) = \{x_2,x_4,x_5,x_6,x_{23}\}$$

$$IL(x_9,h^{\vee},1,2) = \{x_6,x_1\}$$

$$IL(x_6,*,*,*) = \{x_8,x_9,x_{10},x_{15},x_{16},x_{17},x_1,x_{22}\}$$

(b) Let $TL(x,r,y) = \{t \mid r(x,y;t)\}$ be a function with three parameters; x and y are items and r is a relation. Its value is a set of tags which, together with r, form the labels of the links connecting x and y. One of the parameters can be replaced by * to indicate that any relation or item in its position will suffice. Thus, we have the following:

$$TL(x,*,y) = \{t \mid r(x,y;t) \text{ for any } r\}$$

$$TL(x,r,*) = \{t \mid r(x,y;t) \text{ for any } y\}$$

$$TL(*,r,y) = \{t \mid r^{\vee}(y,x;t) \text{ for any } x\}$$

Notice that in the last function the converse relation $r^{\vee}$ is used since the following relation holds:

$$TL(*,r,y) = TL(y,r^{\vee},*)$$

The functions IL and TL are the search functions used to retrieve data from the hierarchy pool as well as the data in the linguistic pool to be described below.

2.4 Linguistic Pool

The linguistic pool is a portion of the data base consisting of a complex network of items. The items in the pool are the constituents of the items in the hierarchy pool. The data structures in the pool represent the syntactic and semantic properties of the items in the hierarchy pool. Figure 2 illustrates the relationship among the linguistic pool, the hierarchy pool, the dictionary and the data base for document texts.

Each item in the hierarchy pool can represent any linguistic unit used to specify the descriptive information of documents. If the linguistic unit is a composite unit, it is analyzed by the system and its syntactic and semantic properties are represented by a dependency structure with semantic labels marking the relations of its constituents. The dependency structure is stored in the linguistic pool. Its corresponding item in the hierarchy pool is connected to the head of the dependency structure through the label (c,t) where c indicates the relation 'constituent' and t is a tag used to distinguish the different dependency structures having the same head. In Figure 2, the item 'record' is used as the head of the dependency structures for 'record of publication', 'record of registration' and 'permanent record'. The tags mark the different dependency structures. The item 'record' is the dependent of the item 'number' which is the head of the dependency structure. The label of the link between 'record' and 'number' is thus (m,1) where m stands for the relation 'modifier'. The relations used in the linguistic pool can be any semantic markers such as 'agent', 'object', 'location', 'membership', 'subset', etc. The set of relations used in the system will depend on the semantic theory adopted by the system.

There exists a many-to-one mapping from the dictionary headings to the items in the linguistic pool. Each item occurs only once in the pool and is represented by a block of machine words just like the items in the hierarchy pool. The data structure of the pool is the same as the hierarchy pool described. Therefore, the same functions defined in the preceding section are applicable for manipulating and retrieving data

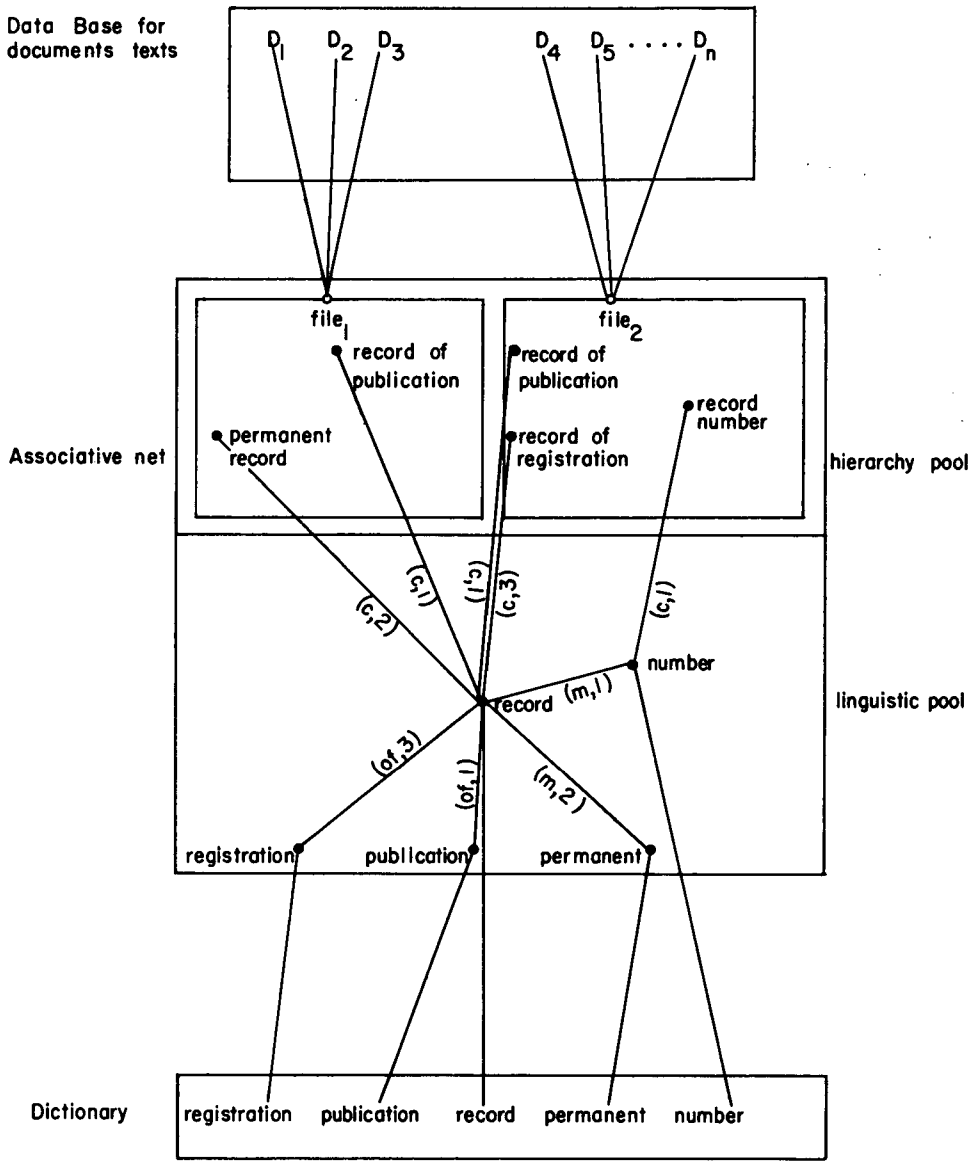


Figure 2. The Relationship Among the Associative Net, the Dictionary and the Data Base for Document Texts

from the linguistic pool.

2.5 A Retrieval Algorithm

To retrieve information from the associative net structure, the following procedures will be followed.

(a) INSEFIRE does a syntactic and semantic analysis of the user's query to determine the dependency structures to be sought in the linguistic pool. This analysis is preceded by a dictionary look-up to obtain the syntactic and semantic properties of the words in the query and to locate the proper items in the linguistic pool.

(b) The linguistic pool is searched by using the search function TL to locate the dependency structures and their heads. From the heads, the system can locate the items in the files stored in the hierarchy pool using the search function IL.

(c) The proper items in the hierarchy pool are located and the structure relations are examined to retrieve specific items or to decode the items above or below the located items in the hierarchy. The same search functions IL and TL are used for this purpose.

The following is a general algorithm for retrieving all records in the associative net structure satisfying a given description. To retrieve information from the data base, INSEFIRE would perform a syntactic and semantic analysis of the query to produce a description. For example, a description may be expressed as the union of a finite number of sets each of which is the intersection of some TL's.

$$(TL_{11} \cap TL_{12} \cap \ldots \cap TL_{1n_1}) \cup (TL_{21} \cap TL_{22} \cap \ldots TL_{2n_2}) \cup \ldots \cup (TL_{\ell 1} \cap TL_{\ell 2} \cap \ldots \cap TL_{\ell n_\ell}) \quad \text{(a)}$$

The TL's in description (a) stand for sets which are the values of the function TL for different parameters. As defined in a preceding section, the search function TL has three parameters: two items and a relation. The items used in the TL's in the description are items in the hierarchy pool. The dependency structures of these items are to be searched first in the linguistic pool. The following steps are followed:

(a) Identify the dependency structures in the linguistic pool: Let $\alpha = I_{111}, I_{112}, I_{121}, I_{122}, \ldots, I_{1n_1 1}, I_{1n_1 2}$ be the sequence of items in the first(next) term in description (a). Items I_{1i1} and I_{1i2} are the parameters of TL_{1i}. Let $\beta = I_1, I_2, \ldots, I_k$ be a subsequence of α containing only the distinct I's. The dependency structure of each item in β is searched in the linguistic pool by using the search function TL. Let

$$TL_1 \cap TL_2 \cap \ldots \cap TL_m \quad \text{(b)}$$

be the description for each item in β. Like in description (a), the TL's in description (b) are sets which are the values of the function TL for different parameters. The parameters in each TL of description (b) are items and relations in the linguistic pool. We can obtain $g = TL_1 \cap TL_2 \cap \ldots \cap TL_m$ as the tag which identifies the unique structure satisfying description (b) and h as the head of the dependency structure. Therefore, searching the dependency structures of all items in β, we will obtain a sequence of tags $g_1, g_2, \ldots, g_k$ and a sequence of heads $h_1, h_2, \ldots, h_k$ of the structures.

(b) Locate the items in the hierarchy pool: The items in the hierarchy pool can be located by the search function $IL(h_i, c', g_i, 1)$ where c' is the converse relation of 'constituent' (see Figure 2). Then, $S_i = IL(h_i, c', g_i, 1)$ for $1 \leq i \leq k$ are the sets of items in different files stored in the hierarchy pool. A sequence of S's can be listed which correspond to the items in α. Let $S_{111}, S_{112}, S_{121}, S_{122}, \ldots, S_{1n_1 1}, S_{1n_1 2}$ be that sequence.

(c) Search the record satisfying a term in description (a): The records satisfying the first term $TL_{11} \cap TL_{12} \cap \ldots \cap TL_{1n_1}$ of description (a) is $J_1 = J_{11} \cap J_{12} \cap \ldots \cap J_{1n_1}$ where $J_{1k} (1 \leq k \leq n_1)$ is the union of all $TL(s_i, r_{1k}, s_j)$ for any $s_i \in S_{1k1}$ and $s_j \in S_{1k2}$. The relation r_{1k} is the relation in TL_{1k} of the first term in description (a).

Repeat the above steps for each term in description (a). The records that satisfy the description (a) would be $J = J_1 \cup J_2 \cup J_3 \cup \ldots \cup J_\ell$.

A simple example would help to clarify the above algorithm. This example will refer to the records in Figure 1. Assume that the user wants to find the document on the subject of information systems written by John Smith. The system will look for the record(s) which satisfies the following description.

TL('subject',a,'information systems') $\cap$
TL('author',a,'John Smith')

In step (a) we have

$\alpha = \beta = I_{111}, I_{112}, I_{121}, I_{122}$ = subject, information systems, author, John Smith.

All the words in the above sequence of terms would be looked up in the dictionary to locate the items (the internal representations of the words) in the linguistic pool. The dependency structures of the terms in the sequence would be identified. Since the terms 'subject', 'author' and 'John Smith' are dictionary headings, they are represented in the linguistic pool by the items 'subject', 'author' and 'John Smith' respectively. The term 'information systems' would have the dependency structure in which the item 'system' is the head and item 'information' is the dependent linked to the head through the relation m(modifier). Thus, the items 'subject', 'system', 'author' and 'John Smith' would be the heads of the terms in the sequence respectively. Assume they all have the tag count 1 which indicates that each item is used once as the head of a depencency structure.

In step (b), the search function IL is used to locate the items in the hierarchy pool. Referring to the items shown in Figure 1, we will have the following:

$$IL('subject',c',1,1) = x_3$$
$$IL('author',c',1,1) = x_4$$
$$IL('John\ Smith',c',1,1) = x_{12}$$
$$IL('system',c',1,1) = x_{13}$$

In this step, we would obtain a sequence $S_{111}, S_{112}, S_{121}, S_{122}$ where S_{111} contains x_3, S_{112} contains x_{13}, S_{121} contains x_4 and S_{122} contains x_{12}.

In step (c), the record which satisfies the description would be

$$J = J_1 = TL('subject',a,'information\ systems') \cap TL('author',a,'John\ Smith')$$
$$= TL(x_3,a,x_{13}) \cap TL(x_4,a,x_{12})$$
$$= \{1\} \cap \{1\} = \{1\}$$

2.6 Remarks

The data structure described in the above sections is believed to be a very general structure that can be used to store a wide range of information. Since the basic structure in the net is an item linking to another item through a label, the number of search functions for manipulating and retrieving data from the net is greatly reduced. The functions TL and IL can be nested to many levels to obtain and decode any datum in the net.

The linguistic pool can be used to store natural language sentences. It is itself a data base for a natural language question answering system. Rules of inference can be stored in the linguistic pool by treating variables as distinct items, and can be interpreted by interpretive routines to deduce answers. The system INSEFIRE under development uses a combination of the techniques employed in document retrieval and natural language question-answering systems. I believe that the proposed data structure is suitable for a system of this nature and is general enough to represent complex information of various degrees.

3. Management of Semantic Data on Disc

In this section, the implementation of the associative net structure and the scheme used for managing the semantic data on disc will be described. The main aim of the data organization and the data management scheme to be described is to facilitate the system to search for a particular item or items that enter specified relations with other items, to collect together sets of items that satisfy specified descriptions and to perform set operations on such sets. All these operations will be done with few references to the disc so that the amount of time required will not become unreasonably great.

3.1 Item Blocks and Link Blocks

The associative net described in the preceding section consists of a complex network of items that can enter into a number of different relations. The number of relations associated with an item depends on the amount of information available which concerns that item and which in turn is entered into the data base. It cannot be pre-determined. If the system reserves a large fixed-length block for each item, too much space will be wasted because some items may only have a few relations. If the system uses a list of single elements for an item, much space will be occupied by the pointers of the list elements. Under these considerations, small blocks of machine words are used to store the information associated with the items. Thus an item is represented in storage by a block (I shall call it an item block) which contains initially a small number of elements (the number is a system parameter and is set to four in the present implementation). If more space is needed for the item, the size of the item block can be increased to accommodate the new information or more item blocks can be established as continuation blocks.

An element in an item block contains a relation and a pointer. The pointer specifies the position of a link block which contains the tags and the addresses of items linked to the item via the labels which are formed by the relation and the tags. Similar to an item block, a link block can be expanded to accommodate additional links or more link blocks can be established as continuation blocks. Thus, two 8-bit fields (element counter and pointer) are reserved in each block (link or item block) to record the number of elements in the block and the pointer to its continuation block. A tag counter is used in each item block to indicate the number of times that the item has been used as the head of dependency structures or as the head of records in the net. The content of the counter plus one will be used as the tag for any structure which is entered into the net and is headed by the item. An item block occupies an amount of storage different from a link block. The following figure (Figure 3) illustrates the structural relation of some items shown in Figure 1.

For the convenience of searching the elements in a list of blocks, the elements in the list are arranged in decending order. The elements in a list of item blocks are arranged according to the internal representation of the relations. The elements in a list of link blocks are arranged according to the internal representation of tags concatenated with their associated item addresses.

3.2 Data Track

Since the associative net will be too large to be held in core, it will be stored in auxiliary storage. In the implementation, the associative net is a regional data set on IBM 2314 disc (7294/track, 312000/sec.). For reasons of operating efficiency a disc track is

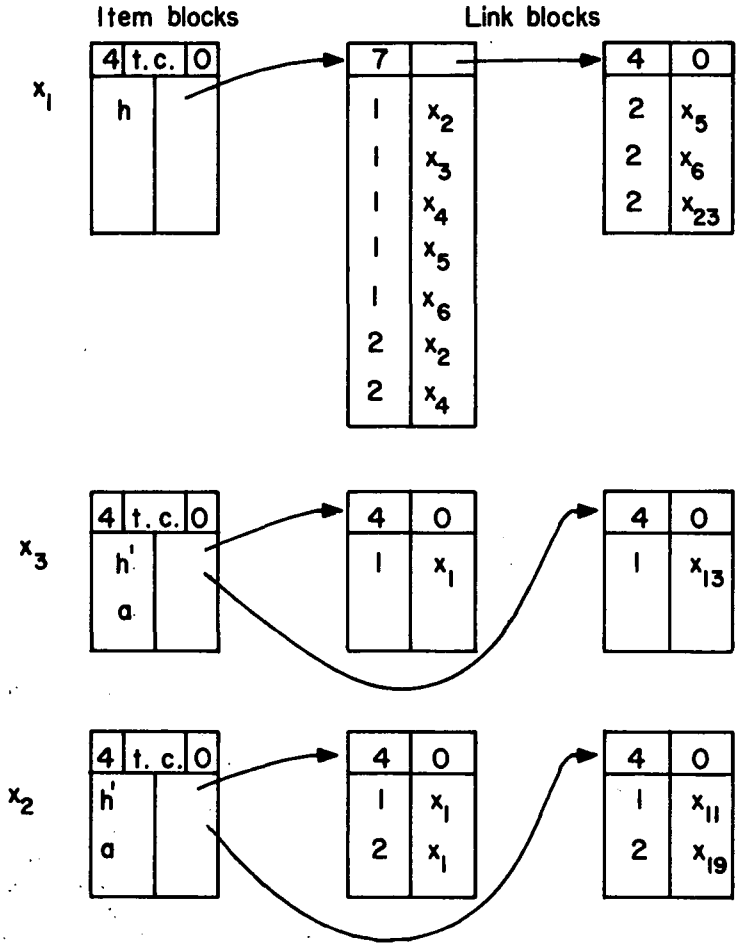


Figure 3. Structural Relations Among Some Items

treated as a logical record. Item blocks and link blocks are established on disc tracks. It is important that the data entered into a track should somehow be related to each other so that once a track is read into core there will be a fair chance that it will provide us with several pieces of useful information. When reference is made to an item, the lists of link blocks which are related to the item are likely to be used. Therefore, it is desirable to store the item and link blocks on the same track. In my design each track is divided into two areas, one holding item blocks and the other holding lists of link blocks. Item blocks are established at one end of a track and link blocks at the other end. The space available for new blocks is in the middle. One advantage of this arrangement is that if the space of one type of block is returned to the available space, it can be used for establishing the other type of block. Another advantage is that as much space as available in a track can be used for establishing either type of blocks. This is important because some information entered into the net may require more item blocks to be established than link blocks. Whereas, other may require more link blocks to be established than item blocks.

The organization of data in a data track is shown in Figure 4.

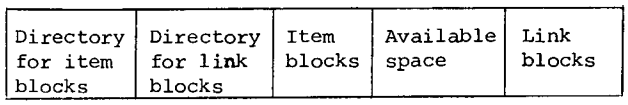


Figure 4. Data Organization on a Disc Track

It is obviously desirable that the removal of a block to another position in a track should not entail changes to references to this block from other items in the net. Each track on the disc will therefore contain a pair of directories. One is used to refer to all the item blocks in the track, and the other for link blocks. An item in the net is therefore identified by a 15-bit track address and an 8-bit offset which refers directly to a position in the directory for item blocks. The entry at that position points to the item block. Similarly, a link block is identified by a 15-bit track address and an 8-bit offset which refers to a position in the directory for link blocks.

The basic operation necessary to enter new information to the net is to add new links to an existing item or link block which, in general, belongs to a list of blocks. The system can store the new links in four possible ways.

(a) store the new links in an existing block, if there is room for them.
(b) the new links may be stored in the adjacent blocks of a current block in a list.
(c) the system may obtain a new block from the AVL space to accommodate the new links and insert the new block in the list.
(d) the system may increase the block size of an old block to make room for the new links.

It is desirable to increase the size of a block rather than to establish a new block because:

(a) to increase the size of an old block will save the cost of establishing a new block, i.e., the space occupied by the continuation pointer and the length of a block.
(b) since an 8-bit offset is used to address the blocks, the system can only establish a maximum of 254 item blocks and 254 link blocks. Increasing the block size would allow the system to use more space in a track.

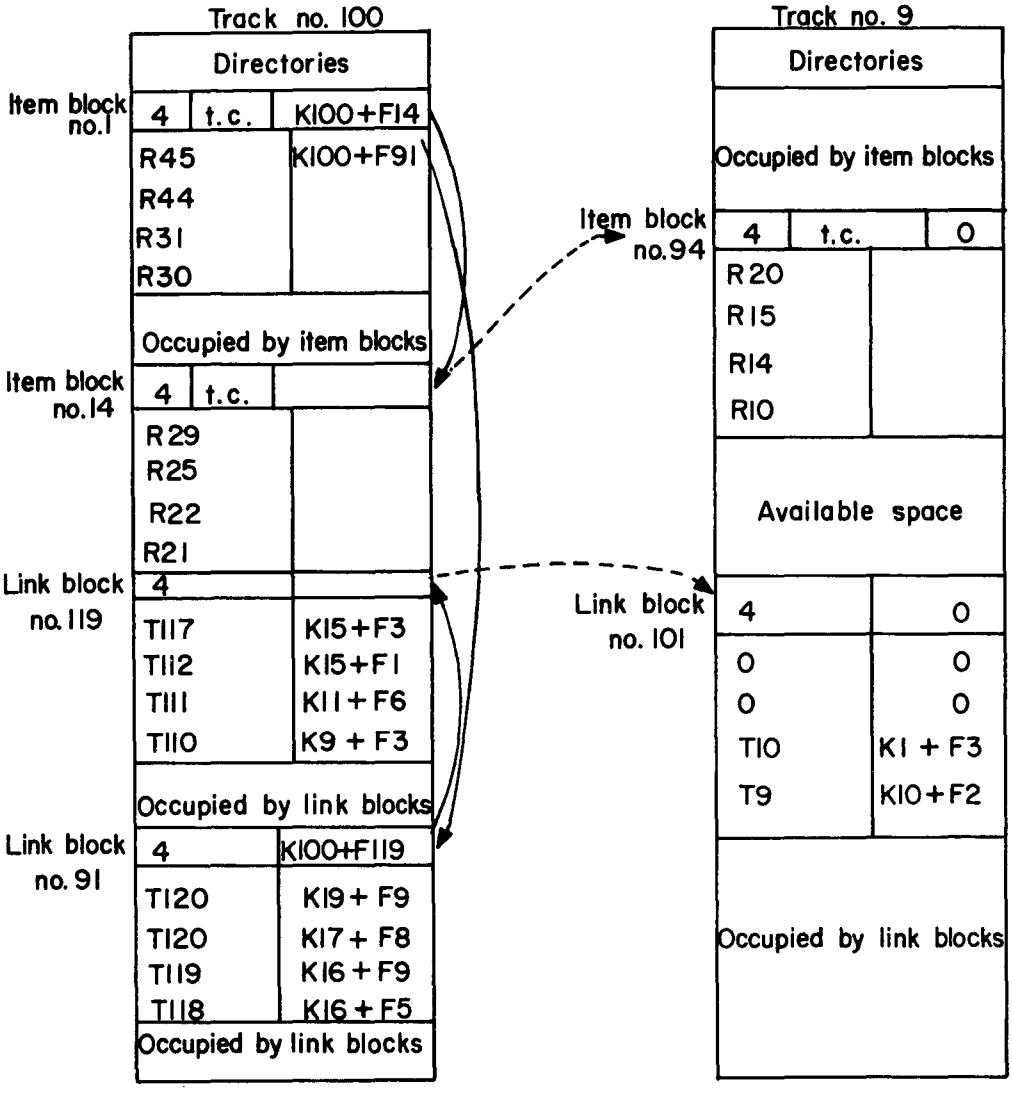
However, we must consider the cost of re-arranging the material on the original track to obtain space which is physically consecutive with the original block. The amount of labor involved in increasing the size of an existing block depends on the number of blocks intervening between it and the available space. A system parameter is used which will be set in the light of experience to determine whether the size of a block should be increased or not.

From time to time, items may be deleted from the net, thus producing pockets of unused space. When the available space in the center of a track is used up, the system does a so-called garbage collection. Only when no further space can be reclaimed is a new track brought into service.

When a link is to be added to an existing list of blocks and no space in a current track is available, a continuation block has to be established on another track. In general, lists of link or item blocks will extend from one track to several other tracks. Information concerning the blocks that continue and the positions of the continuation blocks need to be recorded. One obvious way is to store the position of the continuation block with the block that continues. However, there is a serious drawback in doing so. Since we might have to read several tracks into core to find where the continuation block is located, it is obviously desirable if we can avoid the unnecessary disc access and go directly to the track where information should be or should have been stored.

3.3 Track Continuation Directory

A directory is established for this purpose. It contains continuation information for all data tracks. Each entry of the directory contains the continuation information for one data track. The information is stored in a list of fixed-length blocks. If a list of item blocks (or link blocks) extends from one track to another, an element will be established in the directory. It contains (a) the item address, (b) the last relation (or the last tag concatenated with an item address) in a portion of a list and (c) the position of the continuation portion of the list. Figure 5 illustrates how the continuation information is stored in the directory when a list of item blocks and a list of link blocks in track 100 have



(a) Two Data Tracks

Item address	relation	tag	last element track number	last element offset	continuation block track number	continuation block offset
K100+F1	R45	110	9	3	9	101
K100+F1	0	0	0	R21	9	94

(b) Two Directory Elements

Figure 5. An Example of Lists on More Than One Track

continuation blocks in track 9.

In Figure 5a and Figure 5b, the R's stand for relation names, T's for tags, K's for track numbers, F's for offsets and * is a special marker which indicates that a block has continuations in another track. In Figure 5b, the first element contains the continuation information of the list of link blocks shown in Figure 5a. The second element contains the continuation information of the list of item blocks.

When storing or retrieving a piece of information in the file, the system first refers to this directory to determine the track in which the information should be. Thus, by one disc access, information can be found or stored in the proper track.

The directory may itself occupy several tracks. If we allow the continuation information of a data track to spread over several tracks occupied by the directory, we may have to bring several directory tracks into core storage before we can find the continuation information of a data track. This would destroy the advantage of the directory. The solution to this is to keep all the continuation information associated with each data track in the same directory track.

If the size of the directory is small, it can be kept in core. Otherwise, the directory tracks just as the data tracks will be moved in and out of the core storage. Some approaches are applied to help control the size of the directory.

(a) A continuation block is allowed to be established on a data track only if the track contains a certain minimum amount of space so that data on the track would not shortly be increased and need again be extended to a different track.

(b) The inter-track continuations may be reduced by consolidating the continuation blocks into a minimal number of track when space becomes available.

3.4 Working Space and Track Queue

In this system, a working space is reserved for manipulating data in the net. The working area is large enough for M tracks, where M is a system parameter whose value will be determined on the basis of the core storage remaining after space for programs in the system has been allocated. The data tracks are stored on disc and are brought into core storage when data on the track are to be used or modified. The directory tracks just as data tracks can be moved in and out of the working space. If a track is to be brought into the working space and the working space is full, a track in the working space has to be erased or moved out to disc to make room for the in-coming track. To maintain some kind of priority among the tracks currently in the working space, a queue is established on the basis of how recently the tracks have been used. When a track is used, it is positioned at the head of the queue. A track at the tail of the queue will be removed or erased when space in the working area is needed.

However, it is desirable to keep as many directory tracks in the working space as possible since the system always checks the directory first before using data tracks. A scheme is used in this system to assign the directory tracks a higher priority than the data tracks for remaining in core. When a track is to be removed from the working space, the bottom K pointers in the queue are searched. A data track rather than a directory track will be moved out or erased regardless of the incoming track type. A directory track is to be removed only when none of the K pointers at the bottom of the queue specifies a data track.

3.5 Implementation

A low-level program package has been written and checked out. It consists of 25 PL/1 procedures which establish blocks in both data and directory tracks, manipulate data on disc tracks, search and retrieve data from the associative net. High level retrieval programs can be built upon these low level procedures. The programmer of the high level programs will not have to know the details of the data structure and organization in the associative net.

This program package is a modification and expansion of the work reported in Kay and Su(12).

REFERENCES

(1) Abelson, R. P., and Reich, C. M. Implicational molecules: A method for extracting meaning from input sentences. Proc. Int. Jt. Conf. Art. Intel., Washington, D.C., 1969, pp. 641-647.

(2) Austin, C.J., MEDLARS 1963-1967, National Library of Medicine, Bethesda, Md., 1968.

(3) Bobrow, D.G. A question-answering system for high school algebra word problem. Proc. AFIPS 1964 Fall Joint Computer Conf., Vol. 26, Pt. 1, Spartan Books, New York, pp. 591-614.

(4) Brandhorst, W.T. and Eckert, P.F., Guide to the processing, storage and retrieval of bibliographic information at the NASA Scientific and Technical Information Facility. Documentation Inc., College Park, Md. NASA CR-62033, June 1966.

(5) Charniak, E. Computer solution of calculus work problems, Proc. Int. Jt. Conf. Art. Intel., Washington, D.C., 1969, pp. 303-316.

(6) Colby, K.M., Tesler, L., and Enea, H. Experiments with a search algorithm for the data base of a human belief structure. Proc. Int. Jt. Conf. Art. Intel., Washington, D.C., 1969. pp 649-654.

(7) Coles, L.S. An on-line question-answering system with natural language and pictorial input. Proc. ACM 23rd Nat. Conf. 1968, Brandon Systems Press, Princeton, N.J., pp. 157-167.

(8) Green, C.C. Application of theorem proving to problem solving. Proc. Int. Jt. Conf. Art. Intel., Washington, D.C., pp. 219-239.

(9) Information Storage and Retrieval. Scientific Report no. ISR-16, Cornell University, Ithaca, New York, 1969.

(10) Hillman, D. and Kasarda, A., The LEADER retrieval system. Proc. AFIPS 1969 Spring Joint Computer Conf., Thompson Co., Washington, D.C., pp. 447-455.

(11) Kasher, A. Data Retrieval by computer: a critical survey. Hebrew U. of Jerusalem, Jan. 1966. Tech. Rep. No. 22 to Off. of Naval Res., Inf. Syst. Branch.

(12) Kay, M. and Su, Stanley Y. W., the MIND system: The structure of the semantic file. RM-6265/3-PR, the RAND Corporation, Santa Monica, California, 1970.

(13) Kellogg, C.H. A natural language compiler for on-line data management. Proc. AFIPS 1968 Fall Joint Computer Conf. Vol. 33, Thompson Book Co., Washington, D.C., pp. 473-493.

(14) Kochen, M. Automatic question-answering of English-like questions about simple diagrams. Journal of ACM, Vol. 16, No. 1, Jan. 1969, pp. 26-48.

(15) Lockemann, P.C. and Thompson, F.B. A rapidly extensible language system. Preprint no. 34, Inter. Conf. on Comput. Ling., Sweden 1969.

(16) Longyear, C.R. Memory structures in DEACON natural language questioning-answering system. P-129, General Electric Co., TEMPO, Santa Barbara, Calif., 1966.

(17) Minsky, M. (Ed.) Semantic Information Processing. MIT Press, Cambridge, 1969.

(18) Overmyer, L. Library automation: A critical review. ED 034 107, Dec. 1969.

(19) Quillian M.R. The teachable language comprehender. Comm. ACM 12,8 (Aug. 1969), pp. 459-476.

(20) Salton, G. Evaluation problems in interactive information retrieval. Information Storage and Retrieval, vol. 6, Pergamon Press, 1970, pp. 29-44.

(21) Shapiro, S.C., and Woodmansee, G.H. A net structure based relational question answerer: description and examples. Proc. Int. Jt. Conf. Art. Intel., Washington, D.C., 1969, pp. 325-345.

(22) Simmons, R.F. Natural language question-anwering systems. Comm. ACM, 13, 1 (Jan. 1970), pp. 15-30.

(23) Simmons, R.F. Answering English questions by computer: A survey. Comm. ACM, 8, 1 (Jan. 1965) pp. 53-70.

(24) Simmons, R.F., Burger, J.F., and Schwartz, R.M. A computational model of verbal understanding. Proc. AFIPS 1968 Fall Jt. Computer Conf., Vol. 33, Thompson Book Co., Washington D.C. pp. 441-456.

(25) Su, Stanley Y.W. A semantic theory based upon interactive meaning. Computer Science Technical Report #68, University of Wisconsin, 1969.

(26) Su, Stanley Y.W., Harper K., A directed random paragraph generator. Preprint No. 13, International Conference on Computational Linguistics, 1969.

(27) Walker, D.E. Computational linguistic techniques in an on-line system for textual analysis. Preprint No. 63, International Conf. on Computational Ling., Sweden, 1969.

(28) Weston, P.E., and Taylor, S.M. Cylinders: A relational data structure. Proc. of the ACM Symposium on Data Structures in Programming Languages, 1971. pp. 398-416.

(29) Woods, W.A. Procedural semantics for a question-answering machine. Proc. AFIPS 1968 Fall Jt. Comput. Conf., Vol. 33, Thompson Book Co., Washington, D.C., pp. 457-471.

(30) Yovits, M.C. Information science: Toward the development of a true scientific discipline. American Documentation, October, 1969, pp. 369-376.

A HEATHKIT METHOD FOR BUILDING DATA MANAGEMENT PROGRAMS[1]

Earl Hunt, University of Washington
and
Gary Kildall, Naval Postgraduate School, Monterey, California

ABSTRACT

One of the difficulties faced in implementing information management and retrieval systems is that each case seems to present its own special complexities. As a result information retrieval systems typically fall behind their programming schedule and have many bugs when delivered. In this paper a set of basic operations on types of files are defined. These operations are intended to fulfill the same role for information retrieval systems programmers that functions such as LOG(X) fill for mathematical applications programmers.. they should make the job very much easier. The file operations have been implemented as a run-time package written in FORTRAN IV and Burroughs Extended Algol. The approach has been used to develop three different information management systems; an APL interactive computing system, a generalized information retrieval system, and a specialized information retrieval system for map oriented data. These systems are described.

KEY WORDS AND PHRASES

information retrieval, file management, information management, interactive programming, graphics, systems programming, paging

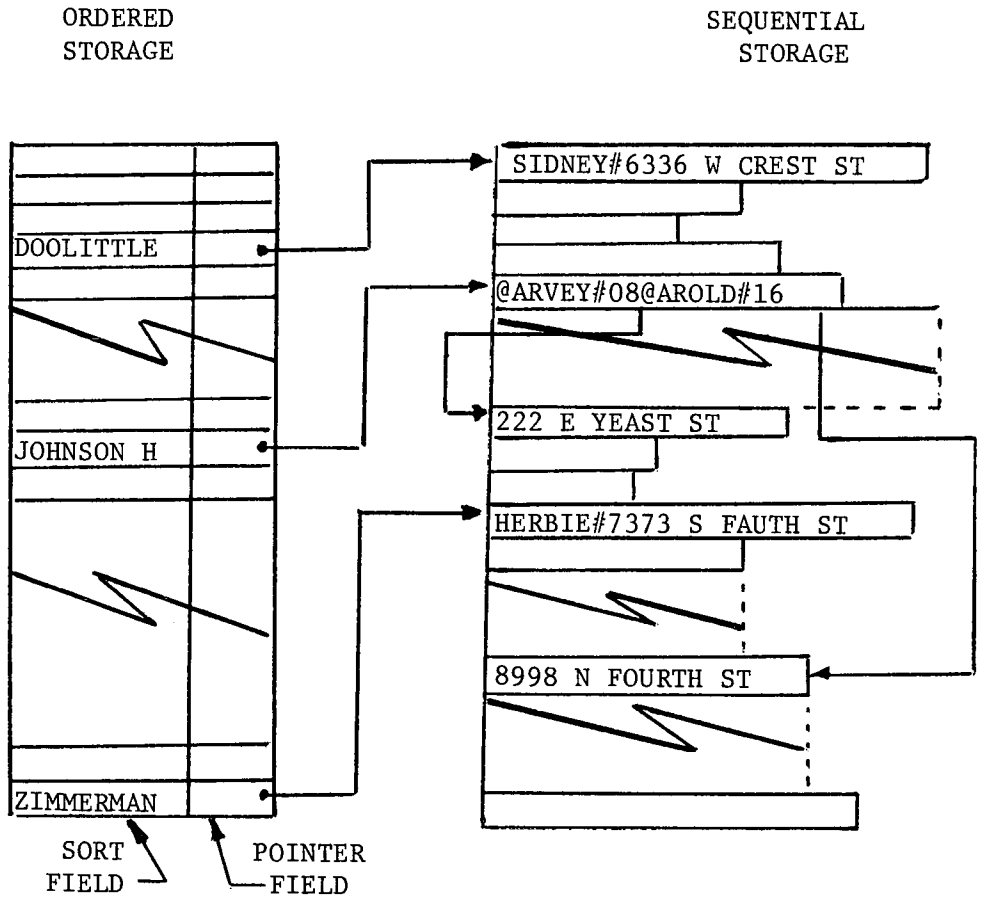
Programs to manage large amounts of intricately structured data are hard to write. They are delivered late and usually contain many bugs when delivered. Only indifferent success has been obtained from "generalized" data management systems. The usual way to write large data management programs is to obtain the services of several wise old (or bright, young) hands and hope that they will read Knuth (7) and Lefkovitz (8) before setting out to reinvent the wheel. Usually this hope is not fulfilled. The resulting code is often so replete with programmer specific or problem specific tricks that it is hard to document, maintain, or modify.

We shall describe a method for the rapid construction of data management programs. The basic idea is that all data management is done through a set of subroutines, called a kernel package which we have found helpful. The programmer using or modifying a system designed with the kernel package need only grasp the few simple concepts and operations involved in the kernel. The approach is somewhat akin to the approach used to build complex gadgetry out of Heathkit or Digibit components, hence the name of the paper. Our approach has been applied in the construction of three substantial applications; a conversational version of the APL language, a generalized information retrieval system, and a system for graphics based information retrieval. At least one other organization has also used our approach successfully. We shall first describe the technique and then discuss the way it was used in each application.

Basic concepts. The term Level 0 storage will refer to word addressable, fast access store (usually core memory), and Level 1 storage will refer to file oriented random access store, normally a drum or disk. We assume a system which has both level 0 and level 1 store. The ideas could easily be generalized to a multi-level system. We also assume a user who writes a program to manipulate data in level 0 store,

1. Portions of this research were supported by the National Science Foundation, Grant NSF B7-1438R, the United States Air Force Office of Scientific Research, Air Systems Command Grant AFOSR 70-1944 and by the University of Washington Institutional Research Fund. During the initial period in which the research was conducted, Gary Kildall held a National Science Foundation predoctoral fellowship at the University of Washington.

FIGURE 1



ORDERED STORAGE PROVIDES ENTRY INTO SEQUENTIAL STORAGE

using the FORTRAN or ALGOL languages, but who has a very large amount of data that is to be held in level 1 store and requested on demand. The kernel package is used to relieve the user from worrying about either the management of level 1 store or the interplay between level 0 and level 1 store. The kernel also provides the user with commands for creating and manipulating very complicated data structures without ever being forced outside of the confines of FORTRAN or ALGOL.

Data structures. The user's data resides in a named file in level 1 storage. The kernel system divides the file in two ways. Physically the file is organized into pages, which are brought into level 0 as needed. The kernel system keeps track of the location of pages in both level 0 and level 1 store. Logically, the user's data is organized into units which he manipulates directly. Insofar as the user is concerned, a unit is a logically connected set of records in one of two types of formats. If the unit is an ordered storage unit its entries are organized by fixed formats called fields. These are established when the unit is created. One of the fields is designated as the sort field. Within an ordered storage unit records are kept sorted on the sort field. Sequential storage units contain variable length character strings, placed in the unit wherever room for them can be found at the time they are inserted. Ordered storage units are typically used to provide indices by which particular strings may be found in sequential storage units. A simple example is shown in Figure 1.

Units are identified to the user's program by unit number. A unit is created (and space reserved for it) only when the user's program calls the appropriate kernel subroutine. Pages are created for units on demand. Page creation and manipulation is done entirely within the kernel system, so the user has no way of controlling or interfering with it.

System use. The subroutines within the kernel system are listed in Table 1. They fall into three major groups. Storage maintainence subroutines communicate between the user's program and the computer's operating system, and declare and release units. They also allow the user a small amount of control in balancing the use of physical resources. The user may specify the amount of level 0 memory to be reserved for pages (thus decreasing the number of references to level 1 storage to get data needed by the user program) and he may specify times at which page and unit assignments in level 1 store are to be re-examined to see if better storage utilization can be obtained by shuffling data between pages. The user does not control the process of examination.

Storage interrogation and alteration commands allow the user to add, retrieve, or delete items to, from, or in units. The items are defined by naming character strings, simple variables, or arrays in the user's program. Finally, utility routines are provided to let the user determine the status of the data created by the kernel system.

The paging method. Let us shift from the user's view of the system as a package of subroutines to the systems program view, as a demand paging system for data.

When the user first initiates the system he names a level 1 file. The kernel divides the file into pages. Each page consists of two parts; the page descriptor and the page contents (user information). The page descriptor indicates the physical layout of user information in the contents area, including the size of the free space area at the end of the page. The unit number and type of the unit to which the page is assigned is also in the page descriptor. Any number of pages may be assigned to a unit. Assignment is done by the kernel system in response to user needs.

The kernel system keeps track of page assignments through the use of an index table in level 0 store. The entries in this table for all pages assigned to a particular unit are stored sequentially. The table entries are descriptors of the page. The relation between the index table and the pages is shown in Figure 2.

TABLE 1
KERNEL SYSTEM SUBROUTINES

Storage Maintenance Subroutines

SUBROUTINE	DESCRIPTION
NAME	identifies the user's level 1 file to the kernel subroutines
SEQUENTIAL	specifies that a particular unit is to be used as a sequential storage unit; unspecified units are treated as ordered storage units
ALLOCATE	indicates the amount of level 0 storage to be used for paging
MAINTENANCE	orders that the assignment of level 1 space be examined by the kernel routines to determine if more efficient balance can be achieved between the pages
WRAPUP	orders that all file maintenance be completed; it is used at the end of the user's program run

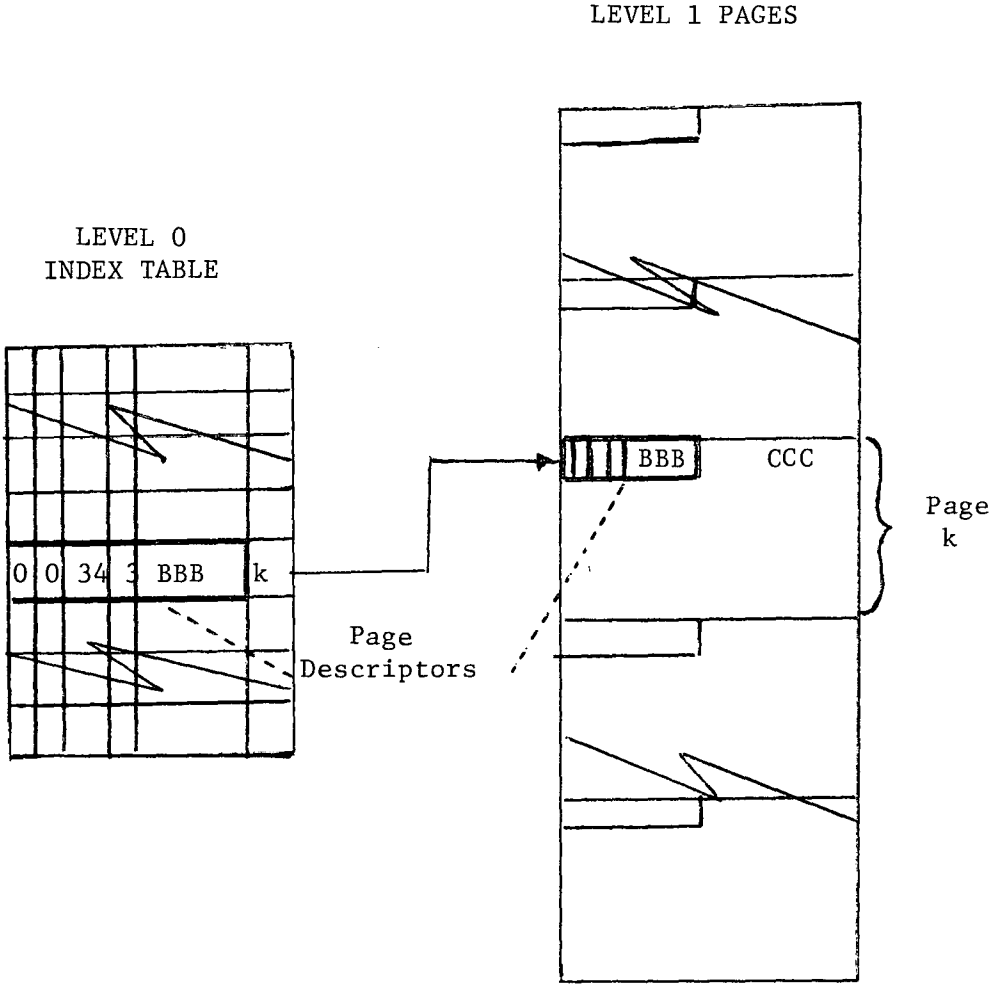
Storage Interrogation and Alteration Subroutines

SUBROUTINE	DESCRIPTION
STORESEQ	stores a character string from the user's program into a specified sequential storage unit
SEARCHSTORE	searches a specific ordered storage unit for a programmer provided character string, and then stores the string in proper sorted order in the unit
SEARCHORD	searches an ordered storage unit for a specified character string; the location of the closest matching string is returned
STOREORD	stores a string of characters into an ordered storage unit in a specific location
CONTENTS	retrieves a character string from a specific programmer provided location in a sequential or ordered storage unit
DELETE	removes entries from ordered or sequential storage units
RELEASE UNIT	orders the kernel subroutines to release all pages assigned to a particular unit to the garbage collector

Storage Utility Subroutines

SUBROUTINE	DESCRIPTION
NEXTUNIT	returns the number of the lowest unassigned unit to the user's program
SIZE	returns the number of entries in a particular storage unit
UNITMODE	returns the mode (sequential or ordered) of a particular unit

FIGURE 2



RELATIONSHIP BETWEEN PAGE DESCRIPTORS AND INDEX TABLE

In addition to the index table the kernel system maintains a unit table which specifies the first and last entries in the index table assigned to each unit. The unit table is also kept in level 0 store.

A basic principle of kernel system operation is that pages should be shuffled as little as possible. Therefore, when a page is assigned to a unit a certain amount of blank free space is left at the end of the page. As more data is moved into the page, the free space is used as a buffer to avoid page overflows. If page overflows do occur new pages are allocated. One of the major jobs of the MAINTAINENCE subroutine is to examine pages, create new pages, and shuffle data between pages to avoid an imbalance between contents and free space that may have arisen as data was manipulated. This is done without moving data from level 1, by using descriptors in the index table. Having a free space area minimizes the number of times maintainence needs to be requested by the kernel system itself. Ideally maintainence is limited to the post run WRAPUP or to times indicated by the user's program. This is particularly useful in conversational application, since MAINTAINENCE can be called while the program is awaiting input from a console. There is no chance of the system becoming unresponsive, for the MAINTAINENCE subroutine (unlike WRAPUP) returns periodically to see if the program which called it wishes to go on to another task.

Copies of one or more pages may be held in level 0 storage at run time. The exact number of copies held is determined by the user, through the ALLOCATE command. The kernel system keeps track of the names of pages in level 0, so that there are no unnecessary accesses to level 1. In general it pays to have at least two pages in core, so that one may have available portions of a sequential unit and portions of an ordered unit serving as an index to the sequential unit.

This concludes our brief description of the kernel program. Kildall (5) describes it in much greater detail. The original kernel was written in Burroughs Extended Algol for the Burroughs B5500 and is inextricably tied to that machine. Two of the applications we will describe are B5500 programs. A second kernel has been written in FORTRAN IV. We have attempted to make it "machine independent" except where it interfaces with the operating system. For instance, the FORTRAN kernel must have the machine word and character size specified to it. The FORTRAN kernel has been tested and used for an application on the XEROX Data Systems Sigma 5, a 32 bit, byte oriented machine.[2] Subsequently we plan to test it on other machines.

Applications

B5500 APL. A conversational APL interpreter has been written for the Burroughs B5500. The system is similar to APL/360 (4) in its outward characteristics. It is a multi-user, conversational computing system operating as a user program under the Burroughs B5500 Master Control Program. B5500 APL is internally divided into several components; a resource management section which schedules work for the other components, a terminal message handler for input and output, a monitor command and function editor section through which the user defines, edits, and traces the execution of APL functions, and a compiler-simulator section which translates from APL code to the order code of an hypothetical APL computer and then simulates the action of that computer. These components are apparent to the user. Quite hidden from him, but of central concern to use, is the virtual memory management section, which controls the allocation of user space in level 1 store. This section is not specialized to APL. It could equally well be used for any conversational computing application where the user needed the illusion of having a very large memory.

2. We are aware of a second FORTRAN kernel used to implement a business information system on a Digital Equipment Corporation PDP-10. We have examined this kernel and, although it is written in Fortran, we feel that it uses so many (quite effective) machine dependent tricks that it is solely a PDP-10 program.

When an APL user enters the system for the first time he is assigned two storage units; an ordered unit called his name table and a sequential unit called his data table. Since the system is designed for multiple users, the numbers of these units cannot be predicted in advance. From the viewpoint of the kernel system, then, the APL system program is the user and the APL user simply an input source for the user program. The APL system calls for units from the kernel as it needs them.

In addition to calling for units when a user enters the system, APL must call for units through its function editor, as the user builds his library of programs and data. The APL user can declare three types of names; scalar names, array names (for numeric or character arrays), and function names. All names, regardless of types, are given entries in the name table. If the name is the name of a scalar variable its current value is also kept in the name table. If the name is an array name, the name table entry contains a pointer from the name table into the data table, where the string of characters defining the array value is located. When the name is the name of an APL function, the situation is more complicated. Upon user declaration of a function two units are created for it, the function label table and the function text table. The label table is an ordered unit containing two types of entries. There is one entry for each line of text in the definition of the APL function definition. The line entry contains pointers to the line definition, which is stored as a character string in the function text table.

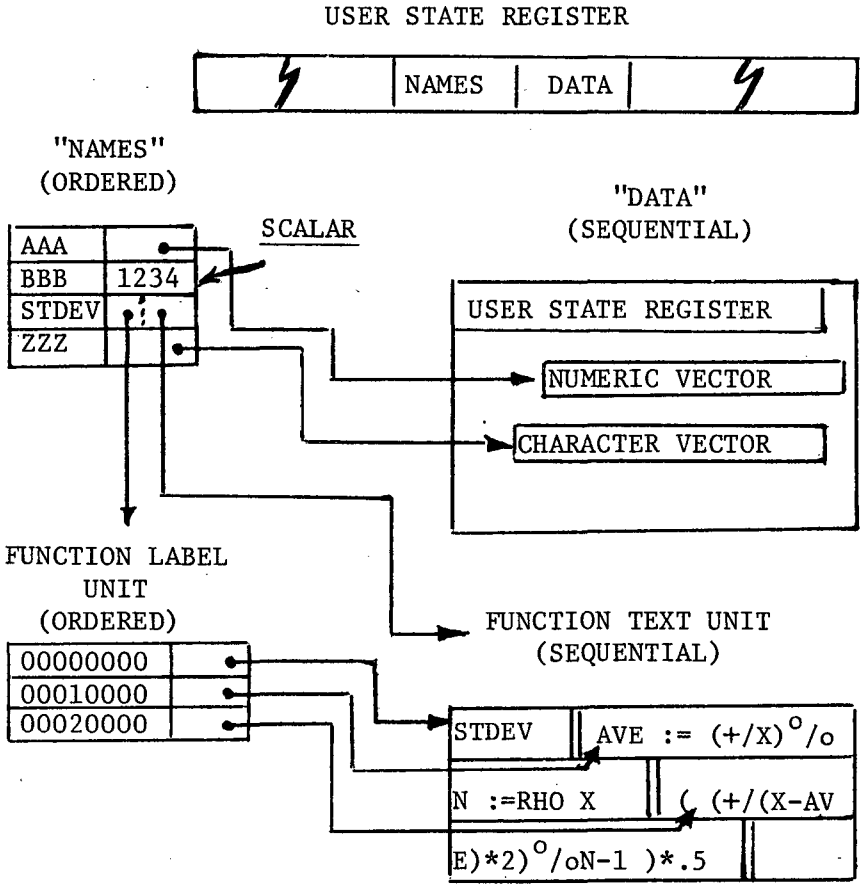
Recall that from the viewpoint of the kernel, the APL system itself is the user, and hence owns the file which the kernel is asked to organize. The result of the above process is that the kernel creates on the APL level 1 file a virtual memory for the APL user, containing his APL program and the value of all his variables. Because the size of the units assigned to a user can be expanded as needed (so long as there is room on the APL file itself) by causing the kernel to create new pages, the APL user has the illusion of having a very large machine. The way this machine is laid out in units is shown in Figure 3. The unit arrangement is transparent to the APL user, just as the page arrangement is transparent to the APL system programmer.

To operate the APL system must have an entry into the units assigned to each user. This is provided by a user state register (shown in Figure 3) which, among other things, contains the unit numbers of the user's name and data table. This is all the system needs to retrieve any piece of information about the status of the user's work area virtual memory. In practice, the user's APL program is executed by bringing appropriate bits and pieces of it from the level 1 file into a level 0 scratchpad area, where data is presented to the APL compiler and simulated APL computer. Note that because of the elaborate entry system into user virtual memory provided by the kernel program, only a very small amount of APL program need be brought into level 0 at any one time. For example, functions may be retrieved one line at a time. This greatly reduces the demand on costly level 0 memory without restricting the size of the program a user may write in APL.

A second interesting result of using the kernel system as a virtual memory for APL is that the system is very hard to disrupt due to computer system crashes. The APL system keeps in its level 1 file the equivalent of a program counter for virtual memory. This program counter is updated whenever new data is moved to or from the scratchpad to level 1. When the machine crashes the level 1 APL file will contain sufficient information to restart the user from the point of the last level 1 access before the crash. Typically, then, the amount of computing the user will lose is measured in milliseconds. This is no small advantage to any interactive computing system.

The B5500 APL system, and with it the Extended Algol kernel program, have now been in operation for over a year. We regard this as a stable application of the Heathkit technique. The details of the B5500 APL system are given elsewhere, Kildall (6).

FIGURE 3



THE FORMAT OF USER 'CORE' STORAGE

A generalized Information Retrieval System. The B5500 Extended Algol kernel has also been used to construct a program called IRSYS, for defining and operating information retrieval systems, Finke (1,2). Like the APL system, IRSYS is best thought of as a program by which the ultimate user defines his application, and not as an application program in itself. Again like the APL application, the IRSYS programmer is quite unaware of the kernel system, although the IRSYS program writer uses kernel continually.

Externally IRSYS looks like any number of other information retrieval systems. Its basic user unit is the data set. A data set consists of a reserved data set symbol, followed by one or more data set elements, and an end symbol. A data set element consists of an element symbol and an element value. The element value may be either a number or a character string. For example, a data set defining a book might be written

```
/DSET
$AUTHOR MEADOW C.T.
$DATE 1970
$PUBLISHER WILEY
$COMMENT TOPICAL TITLE
/END
```

The terms /DSET and /END are the data set symbol and end symbol, respectively. $AUTHOR, $DATE, etc. are element symbols, and the strings following them element values. A data set may have more than one element of the same type. IRSYS is used to define a file consisting of such items and to retrieve items referenced by the values of different elements.

A user interacts with IRSYS in three ways. In definition mode the user states what the reserved symbols will be. Elements are defined to be retrievable with character string values, as $AUTHOR in the example above, numerically retrievable, with numbers as values as $DATE in the example, or miscellaneous, non-retrievable elements ($COMMENT above). IRSYS accepts these definitions and reserves the necessary tables for the user dictionary by calls to the kernel system. Unlike APL, IRSYS is a single user program, so the kernel is used to organize a separate IRSYS file for each user.

In retrieval mode the user writes queries about data sets by expressing Boolean combinations of statements about the values of retrievable elements. Relational statements may be used to reference values of numerically retrievable elements. Thus the query for (($PUBLISHER WILEY) and ($DATE .GT. 1960)) refers to all data sets with WILEY in the PUBLISHER element and with a $DATE element greater than 1960. IRSYS will locate the data sets specified by a query and report their number. It will print these sets only on command, either on the line printer or on a console.

In storage mode the user stores data sets into his IRSYS file and edits data sets already in the file. Obviously one definition session must precede any other session, and at least one storage session must precede the first retrieval session. Otherwise there is no fixed order to the sequence of sessions in each mode. IRSYS can accept input either from a remote console, as an interactive IR system, or from the card reader in batch mode. A file established in batch mode may be interrogated from a console and vice versa.

Now let us look at how IRSYS uses the kernel system. Only a few examples of the technique will be given, since there are many special uses to allow for "pathological" user definitions such as one name's being contained in another. Finke (1) discusses all applications of the kernel with considerable detail and clarity.

Each retrievable element symbol has associated with it an ordered storage unit which is created when the retrievable element is defined. The value entry contains both the value name and a pointer to a second unit in which the strings are the internal numbers of data sets which have the appropriate element and value. Whenever a data set is entered in storage mode it is assigned an internal number. Its elements are then examined. The element name table of each element in the data set is examined to see if the element

value has appeared before. If it has, there will be an appropriate entry in the element table, which will point to a sequential unit. This unit contains a list of previous data sets which have the same element and value as the new data set. The internal number of the current data set is added to this list. If the element value has never appeared before a new entry is made in the element name unit (in the proper place in the sequence) and a new, one entry list is created in the sequential storage unit. The result is a set of cross index tables pointing to lists of data set numbers, as shown in Figure 4. In addition, when the data set is entered its entire text is converted to an internal form (from which the original form is recoverable), and is placed in a sequential unit called the DATA SET Table. An entry for the data set is also made, under its internal number, in the Data Set number table. The Data Set number table is an ordered storage unit whose entries point to the appropriate text in the Data Set Table. The result is shown in simplified form in Figure 4. Not shown is another ordered set in which IRSYS keeps a list of the element names and the number of their associated units.3 IRSYS uses the kernel to keep this list and many others besides. By examining Figure 4 one can see that given an element value, IRSYS has the capability of finding data sets containing that element.

In interactive information retrieval applications the user must often sharpen his question before he locates the set of items he really wants. For example, suppose we were interrogating a file of cinema reviews. The question "$RATING INDECENT" might locate 4970 data sets, while the question $RATING INDECENT AND $DATE .GT. 1965 might find only 100.4 It would be more efficient if the questions were asked in series, and the search for $DATE .GT. 1965 restricted to the set of data set numbers already retrieved in response to the first question. To allow for such situations IRSYS permits the user to "nest" his questions, first asking a question which refers to a set of data sets, and then asking questions which are understood to refer only to that set. The kernel system is used to maintain the various temporary lists needed by IRSYS in this exchange.

IRSYS has now been in operation about nine months without maintainence. It has produced useful results for a number of users.

Graphic Information Retrieval. Our final example is different in a number of ways. The application involves the use of an XDS Sigma 5 to control a graphic display device known as an ARDS.5

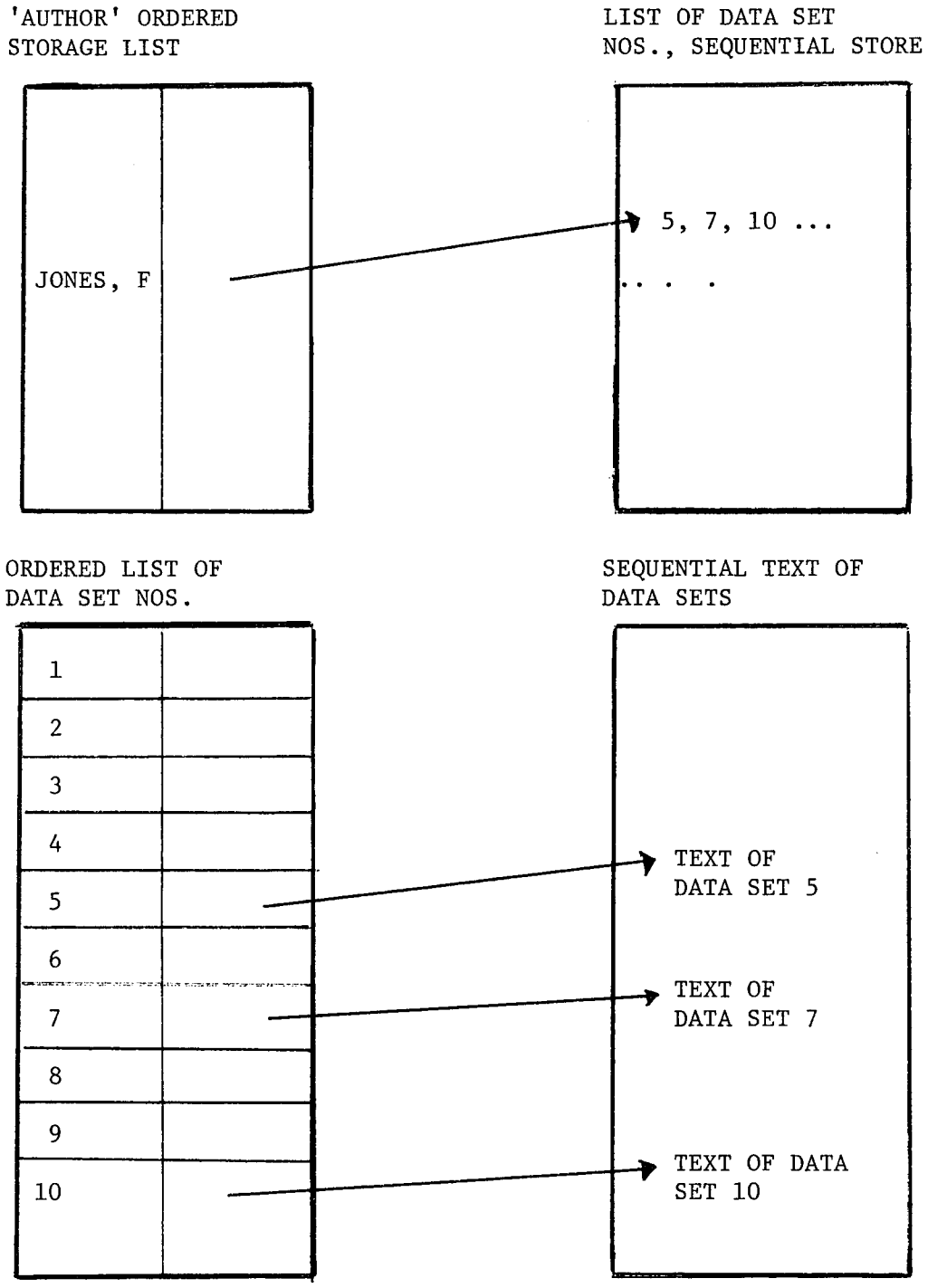
A MAP MANIPULATOR program has been written in Fortran, using a Fortran IV kernel, to allow the user to draw a map (or portions of a map) on the face of the ARDS display. The map is a conventional street map, composed of streets, barriers, areas, and points of various types. Messages may be associated with any map element. Speed indicators are associated with areas and streets. The MAP MANIPULATOR program is not, itself, intended to do anything useful. It is intended to

3. The observant reader may have noted that if this were all that IRSYS did it would be unable to handle situations in which one term included another - e.g. JOHN and JOHNSON, if both terms filled up one of the fixed fields of an ordered storage unit. Such problems are handled in IRSYS by a rather complex system of pointers from ordered to sequential units, the description of which would add little to the current discussion.

4. O tempora! o mores!

5. This is a Computer Displays Incorporated Advanced Remote Display System (ARDS). The ARDS is essentially a storage type CRT, a keyboard, and a graphic input device known as a "mouse". By moving the mouse on a table the user can manipulate a point or vector on the display face, which can then be located by the computer. In addition to the (hopefully) machine independent kernel system this application uses the systems software for controlling the mouse, ARDS, and communication equipment which is part of the operating system of the University of Washington Computer Science Teaching Laboratory, Hunt (3).

FIGURE 4



ORGANIZATION OF DATA SET FILES

IN IR SYSTEM

be a component in a larger system for displaying information in a "command and control" situation in which the user must observe and direct moving units, such as police patrol assignments or air traffic control.

Initially the user "draws" the map on the ARDS display face using the mouse. At any time he may edit the map or associate a message with an element of the map. The user indicates an element of interest either by "pointing" to it with the mouse or by referring to its internal number. (If the user does not know the internal number of an element he may obtain it by pointing and asking.) Once the user has established a map he may ask that different portions of it be displayed, at different magnifications, by zooming or windowing. Thus he has at his command roughly the capabilities of a simplified SKETCHPAD system, Sutherland (9), specialized for map manipulation.

The basic information unit of MAP MANIPULATOR is the MAP TABLE. This is a sequential unit whose records are strings in a language for defining map elements. Each sentence in this language must conform to a BNF syntactical specification. For example, the grammatical definition of a road is

<road>::= 1 <roadname> <route> | <road> <route>
<roadname>::= {positive integer}
<route>::= <speed> <line>
<speed>::= {negative number}
<line>::= {co-ordinate pair} {co-ordinate pair}

The associated semantics are:

1 is an identification symbol to aid in interpretation. Remember that this "language" is for strings that will be read by a program, not a person.

The <roadname> integer is a pointer to an entry in the ROAD TABLE, an ordered unit that will be described below.

The absolute value of <speed> indicates a multiple of the basic speed. This is interpreted as the value of speed of movement along the road. It is intended for use in answering questions about best routes from one point to another, or about the transit time along a particular route.

The co-ordinate pairs for the <line> are the endpoints of a vector. In the MAP TABLE they are in map co-ordinates, i.e. they refer to the scale of the map, which may be much more than can be displayed.

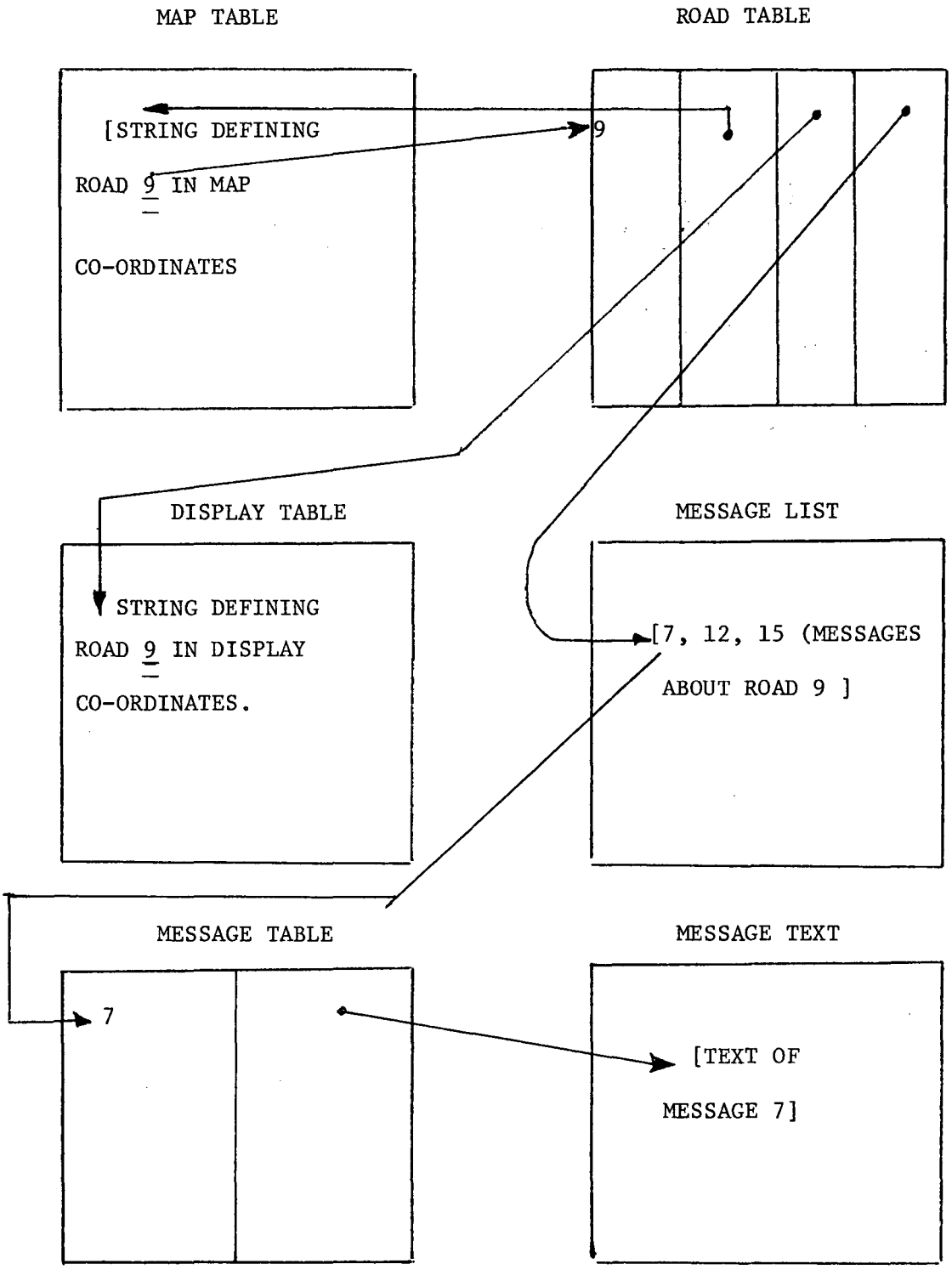
ROAD TABLE is an ordered unit which contains one entry for each road in the map. This entry contains pointers back to the string defining the road in the MAP TABLE, and, if appropriate, to the string defining the road as displayed (in display face co-ordinates) on the sequential unit DISPLAY TABLE. In addition, a ROAD TABLE entry contains a pointer to a MESSAGE LIST table entry. MESSAGE LIST is a sequential unit whose records are lists of message numbers associated with a particular map element. Each message number names an entry in the ordered MESSAGE TABLE, which, in turn contains a pointer to the message text in the sequential MESSAGE TEXT table.

The overall structure of the data defining roads in MAP MANIPULATOR is shown in Figure 5. Similar figures could be constructed showing tables for barriers, areas, pointers and events.

We have pointed out that MAP TABLE defines elements in terms of map co-ordinates. A DISPLAY TABLE unit, similar in form to MAP TABLE, defines the elements actually being displayed at any one time in terms of display co-ordinates. This is necessary both for windowing and zooming and to determine what element the user is pointing at with the mouse.

The structure of data in MAP MANIPULATOR gives us the potential for asking a number of questions about graphically displayed data. For example, one can display a map, indicate an area by pointing at it, ask that the area be displayed at an appropriate magnification, and either insert or retrieve messages about the area. Note that these messages could be entries into an information retrieval system similar to one established by IRSYS. We have not actually made this connection as yet.

FIGURE 5



STRUCTURE OF MAP MANIPULATOR

Summary

Nothing we have reported is terribly exciting or new. Building complex systems requires sophisticated tools. The Heathkit approach is an attempt to make it easier to program complex information management systems, and to produce programs that are easy to maintain and understand. Basically, the approach will work if many systems can be designed with the same tools. It will not work if every information management problem is unique. We think the Heathkit approach works well. While we certainly do not want to disparage any of our colleagues who have worked on the various applications we have described, we think it is correct to say that none of them came to these projects with any substantial background in system programming.

The timetable for the results was as follows:

B5500 APL, which includes many things besides the virtual management memory package, was completed by four people, working at about a 25% effort for six months.

IRSYS was completed by one person working half time for six months.

MAP MANIPULATOR was brought to a nearly workable stage by two people working quarter time for three months. A substantial part of this time was spent checking the Fortran IV kernel system. The kernel system is far from simple, and obviously must work if anything else is to work.

We believe this is a good record. It may be, of course, that the resulting programs execute very inefficiently. It is hard to determine whether inefficiencies should be blamed on the Heathkit approach or on the fault in a particular implementation. In fact we are not at all sure that inefficiency does follow from our approach. IRSYS has been used by a number of people who have not complained about the bills. IRSYS contains routines to monitor system operation, and this is currently being done. APL B5500 appears to be reasonably economic if the system on which it runs is not too heavily loaded. In any case, execution efficiency is not the only criterion for system programming. There is a good argument for bringing up a working system quickly and then replacing it with a highly efficient one at your leisure. How many microseconds are there in a month?

Acknowledgements

So many people have aided us at various stages that our acknowledgements must perforce read like movie credits. We list them alphabetically by project.

APL project. Hellmut Golde, LeRoy Smith, Sally Swedine, and Mary Zosel.

IRSYS. Jeanne Finke

FORTRAN IV Kernel. Thomas Kuffel

MAP MANIPULATOR. Jerry Jensen and Duane Sand.

And last, but not least, let us mention the first user who was willing to rely on a Heathkit built system (IRSYS) to solve his problem, although he did not have the slightest interest in advances in computer science. Such a user always discovers the bugs in the perfectly checked out system. Our user, Professor David Bonsteel of the University of Washington School of Architecture, was no exception. May his buildings never crash.

REFERENCES

(1) FINKE, JEANNE M. A generalized file management system. U.W. Comp. Sci. M.S. thesis, (1970a).

(2) FINKE, JEANNE M. A user's guide to the operation of an information storage and retrieval system on the B5500 computer. U.W. Comp. Sci. Tech. Report 70-1-3, (1970b).

(3) HUNT, EARL, The computer science teaching laboratory at the University of Washington. ACM, SIGCSE Bull. 2, 1970, 30-33.
(4) IVERSON, K. and FALKOFF, A.D., APL/360: User's manual. IBM Corp., 1968.
(5) KILDALL, G.A., Experiments in large scale computer direct access storage manipulation, U.W. Comp. Sci. Tech. Report 69-01, 1969.
(6) KILDALL, G.A., APL/B5500: The language and its implementation. U.W. Comp. Sci. Tech. Report 70-09-04, 1970.
(7) KNUTH, R.D.E., The art of computer programming. Vol. 1 Menlo Pk., California Addison-Wesley, 1969.
(8) LEFKOVITZ, D., File structures for on-line systems. New York Spartan, 1969.
(9) SUTHERLAND, IVAN E., SKETCHPAD, a man-machine graphical communication system, Proceedings of the Spring Joint Computer Conference, 1963, Spartan Books, New York pp. 329-346.

FILE STRUCTURE DETERMINATION

Anthony J. Winkler, United States Air Force Academy
Alfred G. Dale, The University of Texas at Austin

ABSTRACT

An approach to determining an appropriate file structure for a given application is presented, by outlining a methodology for comparing some important aspects of data management system performance. The aspect chosen for analysis is the processing time required to evaluate Boolean functions defined on data values contained within a file structure and select elements from the structure satisfying the expression.

Two file structures are studied. The structures are each combinations of hierarchical and inverted file organizations, which differ in the use of the pointers contained in the inverted file. In one case they link a value to nodes corresponding to its occurrence in the data hierarchy and in the second they link a value to the entry which contains the node corresponding to an occurrence.

Algorithms for processing within each of the structures are discussed. Each algorithm is then modeled, and approximating models developed for simulation of the algorithms.

KEY WORDS AND PHRASES

file structure, data management system, data structure, storage structure, processing algorithm, modeling, simulation, Boolean function evaluation, retrieval, update, hierarchical file, inverted file, hierarchy, logical entry, file structure determination.

I. Introduction

A data management system consists of a set of file structures, and a set of algorithms for (a) creating the structures and (b) selecting data from the structures for output or for modification. A wide variety of file structures exist, and a major problem in the design of a new system, or the selection of an existing system, is the determination of appropriate file structures for a given application.

The purpose of this paper is to discuss an approach which may aid in the solution of this problem, by outlining a methodology for comparing some important aspects of system performance. The aspect chosen for analysis is the processing time required to evaluate Boolean functions defined on data values contained within a file structure and select elements from the structure satisfying the expression. These processes form the most important part of data management algorithms required for retrieval and updating operations.

The technique is an aggregated one in that it does not explicitly separate data-independent influences on processing from data-dependent factors. Algorithmic performance is a function of three sets of variables: data structuring techniques used by the system; the specific configuration of a particular data base; and the composition of the Boolean function. The latter two are data-dependent, and the first is data-independent. Current work is in progress at the University of Texas at Austin in the development of models designed to isolate the influence of file structure on processing algorithms as an extension of the work reviewed in this paper (1).

The methodology requires that a model of each processing algorithm be

Preceding page blank

developed. This model is in the form of an equation, the left side of which is the processing time, while the right side consists of those parameters which influence the processing time. Once these parameters have been determined, the equation for each structure's processing algorithm can be simplified to a model which contains only those parameters significantly influencing the processing time. This simplification allows the processing algorithms to be simulated by varying the value of the important parameters. Examination of the processing times which result shows those transition points where one structure begins to require more time than another. With these results one can then examine the processing algorithms to determine whether the results were biased by any initial assumptions. If so, the assumptions may be changed and the process repeated.

This methodology can be used for at least two important purposes. The first is to determine that storage structure which best satisfies a given set of requirements. The second is to identify the areas of the processing algorithm which are inefficient, and those factors which most influence the processing time. With this information either more efficient algorithms can be developed, or variations on the file structure can be implemented, or both.

The methodology is similar to the procedure used by Martin (6), whose work was based on that done by Lefkovitz (4). Lefkovitz analyzed five different file structures, ranging from multilist to inverted file. Comparisons were made by first examining a particular file structure, and then noting its advantages or disadvantages over another in particular circumstances. Martin attempted to develop a single model which could be used to compare three of the file structures developed by Lefkovitz. Had he been successful, he could have compared the file structures to determine which was best for a given set of values of the parameters included in the model.

The technique used in this paper differs from the two above, in that a detailed processing algorithm is defined for each file structure, these algorithms are then modeled, and the results are compared.

II. The Methodology

To illustrate how this methodology can be used, two selected file structures are examined. Similar studies have been made previously of other structures (7). Each structure is a combination hierarchical/inverted file but differ in the composition of their inverted files. The reason for choosing to compare these structures is that one represents that used in an operational data management system (2,3) and one is a proposed modification.

III. Data Structure

Each file structure to be discussed derives from a data structure definition, which specifies hierarchical relationships among data elements. An example of a data structure definition for a data base named Student Records (7) is presented below.

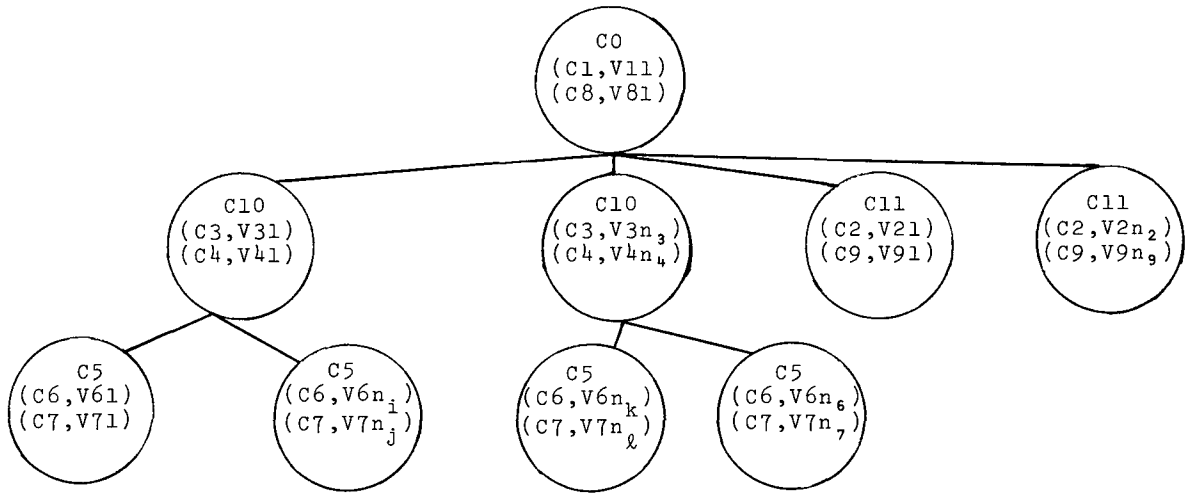
```
 1)  Student Name (ELEMENT)
10)  Academic Record (REPEATING GROUP)
       3)  Semester (ELEMENT in 10)
       4)  Semester Hours (ELEMENT in 10)
       5)  Courses (REPEATING GROUP in 4)
             6) Course Title (ELEMENT in 5)
             7) Grade (ELEMENT in 5)
 8)  Sex (ELEMENT)
11)  Graduation (REPEATING GROUP)
       2)  Data (ELEMENT in 11)
       9)  Degree (ELEMENT in 11)
```

The definition is a representation of the data structure of the file named Student Records. It is given in terms of the attributes Student Name (C1),

Academic Record (C10),..., Degree (C9).

An element is an ordered attribute/value pair (element name, element value). A data set is either an ordered set of elements comprising the contents of a node, or the null set. A node is an ordered pair (repeating group name, data set). A repeating group is a set of nodes such that each node in the set has the same repeating group name, and each node has the same parent node. A logical entry is the highest level subtree which is not the entire file, and which contains zero or more repeating groups below its topmost node.

A particular instance of the organization of elements into related data sets satisfying the definition schema could result in the tree:



This tree is the basic structure for a logical entry. In the tree, C0 refers to the top node of the logical entry. The ordered pairs within each node represent the elements contained in the node. The V_{i_j} represent the j-th value for the i-th element.

IV. File Structures

Representations of the file structures to be discussed are presented in Figures 1 and 2. The file structure is the internal representation of the data structure. That is, the file structure contains all pointers, element names, values, etc., required for processing within the structure.

The first structure (S1) is a combination of both hierarchical and inverted file structuring techniques (see Figure 1). For the hierarchical structure

Figure 1

An Example of S1 for a Logical Entry

Portion of the inverted file for this logical entry:

C1	V11	P11
C2	V21	P21
⋮	⋮	⋮
C2	$V2n_2$	$P2n_2$
C3	V31	P31
⋮	⋮	⋮
C3	$V3n_3$	$P3n_3$

C4	V41	P41
⋮	⋮	⋮
C4	$V4n_4$	$P4n_4$
C6	V61	P61
⋮	⋮	⋮
C6	$C6n_6$	$P6n_6$

C7	V71	P71
⋮	⋮	⋮
$C7n_7$	$V7n_7$	$P7n_7$
C8	V81	P81
C9	V91	P91
⋮	⋮	⋮
C9	$V9n_9$	$P9n_9$

Hierarchical Structure for this Logical Entry

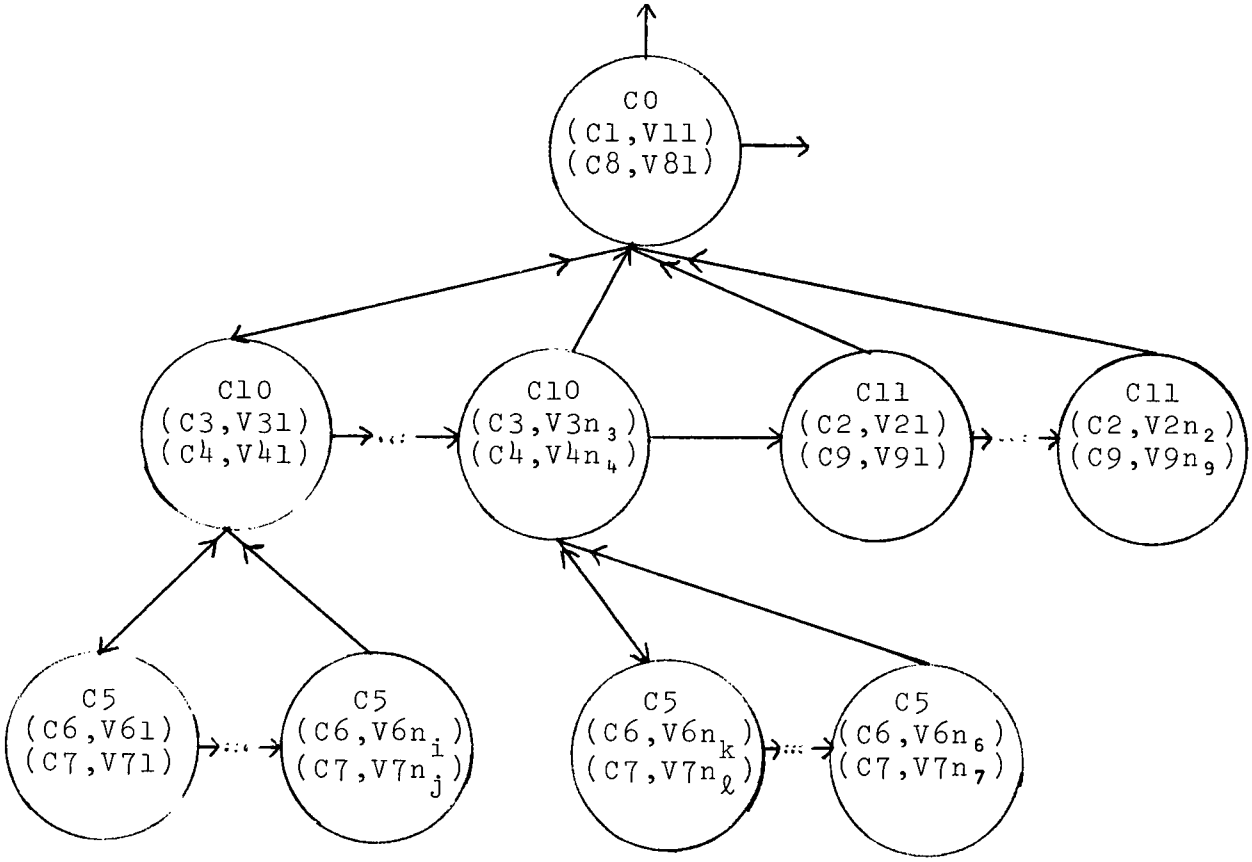


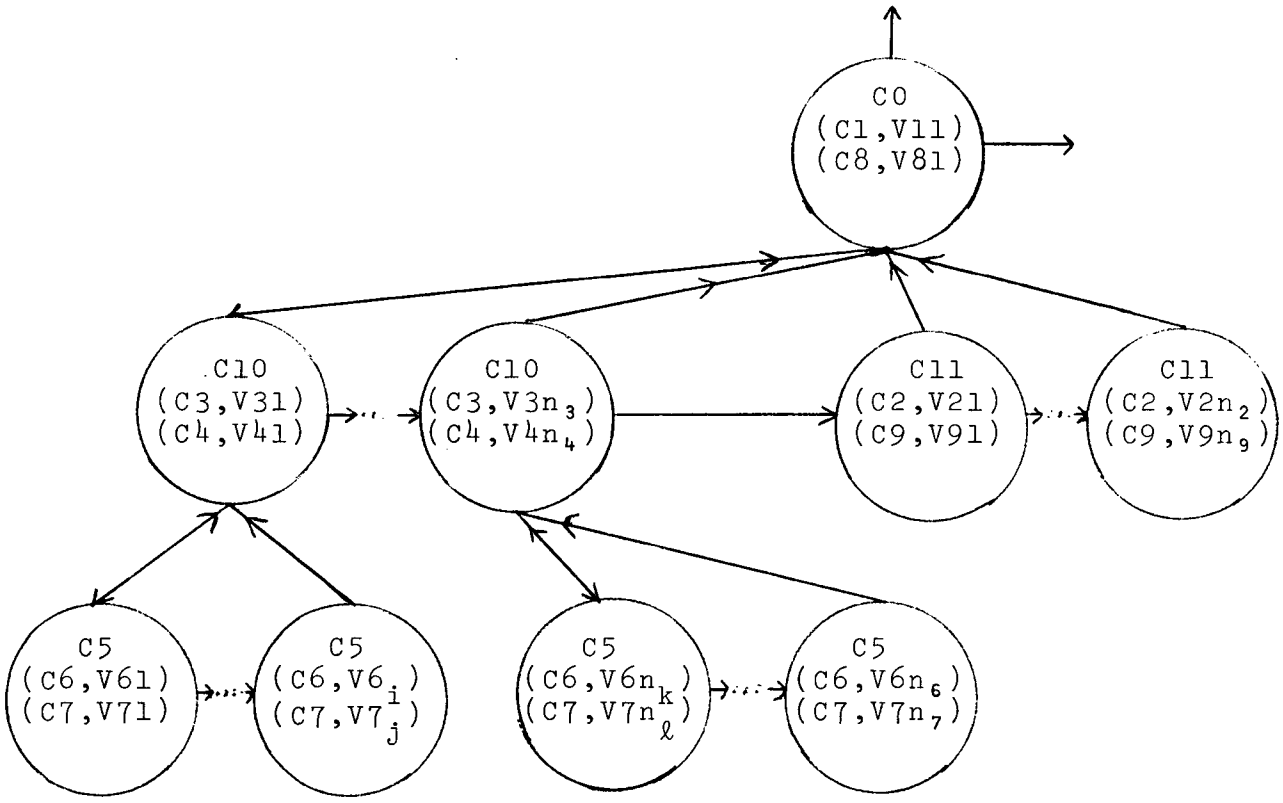
Figure 2

An Example* of S2 for a Logical Entry

Portion of the inverted file (see Figure 1) for this logical entry:

C1	V11	P0
C3	V31	P0
⋮	⋮	⋮
C9	$V9n_9$	P0

Hierarchical Structure for this Logical Entry



*The elements within the nodes are the same as those for Figure 1.

the hierarchy is triple-chained with pointers up, down and to the right for each node. The inverted file associates each element value with every location in the hierarchy where it occurs, that is, the nodes of the tree in which the value occurs. This combination structure is illustrated for a single logical entry in Figure 1. For the hierarchical structure in the figure, the pointers are indicated by arrows, and the group names and element names are those used in the data structure definition.

For the inverted file structure in Figure 1, each Pij is the location of the element value, Vij, in the logical entry hierarchy.

The second structure (S2) is a proposed hierarchical/inverted file structure. In this inverted file the pointers associated with an element point to the logical entry which contains the node associated with the element rather than to the node itself. If the same element value appears more than once in a logical entry, there is only one pointer to the logical entry from the inverted file. This structure is illustrated by Figure 2.

V. Processing Algorithms

With respect to each file structure, algorithms can be defined which implement a statement referred to as a "request statement." This statement has the form, "Action X where Q." The processing of a request statement involves two steps. First, the Boolean query Q is processed. This results in a sequence of subtrees from the data base. Second, the subtrees are processed sequentially, according to the command specified by Action for each element of X. Interrogation is defined by Q; retrieval and maintenance are defined by Action. For the right-side of the request statement, the Boolean query is assumed to be in disjunctive normal form, without complements. Thus, a query has the form of a sequence of terms "OR"'d together. Each term is a sequence of operands "AND'ed together. The operands of the query are conditions on values of components. Examples of a query on a student record file are

(a) (STUDENT NAME = WINKLER);
(b) (COURSE = CS 699) AND (GRADE exists).

Then, (a) has the value true when the student's name is Winkler; and (b) is true if the course is CS 699, there is a value for grade, and as for this example, both values belong to the same data set. Although only the relational operator "=" is used in the above examples, any standard relational operator is allowed. Also allowed is the "existence of a value" operator, signified by "exists." The symbols (and) are used to delimit the condition.

The algorithms, designed to process the various configurations of a request statement will not be detailed in this paper. They may be found in (7). However, certain observations will aid in understanding the models of the algorithms discussed in the next section.

The algorithms have some overlapping processes, and some important differences. For S1, the "right-side" of the request statement is processed using the inverted file. The "left-side" is processed using the three-pointer hierarchical structure and the inverted file structure for maintenance. For S2 the right-side processing is two-fold. In the first case, the procedure is similar to the right-side processing of S1, except that sorting and comparing is on the logical entry designator. In the second case, only the logical entries selected by the above preprocessing require processing within the hierarchical structure.

For each of the algorithms several assumptions are made. The first set of assumptions eliminates those procedures which are algorithm independent. First, it is assumed that the Boolean expression has been edited, the NOT operators have been eliminated, and the expression is configured in disjunctive normal form. The NOT operator can be eliminated by substitution of the negation of that relational operator associated with the operand. (The negation of "existence of a value for repeating group" is not considered since it requires special processing unique to using the inverted file for Boolean processing and thus effects both structures similarly.)

The second assumption is that the operands in the Boolean query are ordered within each term and then termwise in the order specified by the data structure definition.

The third assumption is that the relative position of an element or group (within the data structure definition) is associated with each element name or group name involved in the Boolean query. As the Boolean query is parsed, this last requirement eliminates a table search in the processing algorithms.

The second set of assumptions is concerned with the use of internal and external memory in processing within each of the storage structures. In the case of S1, as in RFMS (3), tables are used to represent the two storage structures. Two of these tables are considered to be in internal memory at all times that the data management system is being used. These are the data structure definition table and the directory to a sorted table of unique values for each element. The remaining tables are considered to be on an external memory device. The first table consists of all unique values occurring for each element, the element name and a pointer to either the location of the value (in the hierarchy), if it occurs only once for an element value, or to the first location in a second table consisting of lists of pointers. Each list in the second table corresponds to the location (in the hierarchy) of each of the duplicate occurrences for a unique value. The third table represents the hierarchical structure. Each node of the hierarchy is represented by an entry in this table. Each node represents a data set. The location of the node is represented by three pointers (UP, DOWN and RIGHT). The value(s) of a node is represented by a set of pointers to the appropriate location(s) in the table of unique values. It is assumed that all nodes associated with a logical entry are accessed from external memory when one node is accessed. After processing, the entire logical entry is returned, as a block, to auxiliary storage. Each of the three tables may contain additional information for each entry. This information, such as level and component name, can be used to aid processing, but is peripheral to the main contents of each table.

For the table of unique values, it is assumed that if a unique value of an element is accessed from external storage, a block (partition, page) of unique values of the element are accessed. These values will be available for processing until a unique value for another element is accessed from external memory. For the table of pointers to duplicate occurrences, all pointers corresponding to a particular value of an element are accessed from external memory when one pointer is accessed. These pointers are available in internal memory until pointers corresponding to another value are accessed.

It is assumed that each value is accessed from external memory as it is needed. This assumption affects each algorithm equivalently when operating in

TABLE 1

Parameter Definitions

Symbol	Definition
ADI	- the average number of duplicate values for all C_i in the Boolean query.
AE	- external (auxiliary storage) memory access time.
AI	- internal (core) memory access time.
ANDS	- the expected number of ANDs in a term of the Boolean expression; represented by PI @ N/2 if PI < 1.0, or N if PI = 1.0.
K	- the average number of data sets satisfying an operand of the Boolean query for all operands; represented by LE@NDIS.
LE	- the number of logical entries in the file.
MEM	- the amount of internal (core) memory available for internal sorting.
Mi	- the processing algorithm for Si, i = 1, 2.
N	- the number of operators in the Boolean query.
NDI	- the average number of nodes per logical entry for all attributes in the Boolean query.
NDIS	- the average number of elements satisfying a given query on a logical entry for all attributes in the Boolean query.
NUQCHN	- the number of unique subtrees satisfying the Boolean query; represented by LE@NDIS@(N+1)@ (PU+PI@PTRACE).
NVI	- the average number of values for all components in the Boolean query; represented by LE@NDI.
NXI	- the number attributes referred to in the left-side of the request statement.
PADV	- the probability of additional values existing for a component, having already processed one value.
PBI	- the ratio of the expected amount of processing time allocated for BI processing in the hierarchy to the total amount of section B processing required, i = 1,2,3; PB1 + PB2 + PB3 = 1.

B1 processing occurs when the left-side (X) of the Boolean query contains an attribute which is located within the subtree satisfying the right-side (Q) of the Boolean query.

B2 processing occurs when the subtree is in the down-trace of an attribute in X.

B3 processing occurs when neither B1 nor B2 occurs.

PDP - the probability of existence of duplicate values for a unique value of the component.

PI - the proportion of ANDs expected in the Boolean query.

POC - the probability of existence of a value when a value is being added to the file.

PS - the partition size.

PSAT - the probability of satisfying the Boolean query; represented by NDIS/NDI.

PTRACE - the probability of additional processing being required by the AND operator; represented by PI@(PSAT to the ANDs power).

the hierarchical structure.

VI. Model Development

Models embodying the algorithms will now be discussed. The purpose in modeling is to be able to simulate the algorithms under varying conditions. The models are programs representing the required processing of each algorithm. Following the presentation of the basic models is a discussion of the development of the simplified models used for simulation. The simplified models are not presented since they can be easily derived from the basic models and the assumptions regarding the variables. In the development of the two models, only those parameters which involve external accesses are considered. This is done for purposes of simplification and because of the negligible effect of internal storage accesses on total processing time.

In the interest of readability, the parameters in the models are abbreviated. For ease of reference, these abbreviations are listed in Table 1.

VII. Model 1 (Processing in S1)

For M1, the total processing time is based on the processing of the right-side of the request statement (RTIME), the left-side processing for the hierarchical structure (LTIMEH), and the left-side processing for the inverted file structure (LTIMEI).

```
RTIME = (N+1)@AE@(2+(NDIS/ADI)@LE@PDP
        @(1+ADI MOD PS)
        +((NDI-NDIS)/ADI)@LE@PADV@
         (NVI/ADI MODULO PS)))
        +AE@(((N@(K MODULO MEM)MODULO PS)+1)
        +Maximum(((K-MEM)MODULO PS),0))
```

```
LTIMEI = NUQCHN@NXI@M@AE@(3-POC+PDP
        +((((NVI/ADI)+1)/2)MODULO PS)).
```

```
LTIMEH = NUQCHN@NXI@AE
        @(PB1@(PI@PTRACE+PU)+PB2+PB3@NXI
        +M+.1).
```

Thus, the total processing time for M1 for a given request statement is given by

TTIME = RTIME + LTIMEH + LTIMEI.

VIII. Model 2 (Processing in S2)

For M2, the total processing time is dependent on the procedures defined by the pre-process section of M1, and the right-side and left-side sections of the hierarchical processing section. This algorithm uses the inverted file to narrow the number of logical entries to be processed, and does this by selecting only those logical entries which can possibly be qualified by the Boolean query. TLE is the number of such logical entries which are qualified. For this model, TLE is represented by TLE*(PI*PTRACE+PU), where LE is the total number of logical entries in the file.

The model for the total processing time for M2 is

```
PRETIME=(RTIME for M1 with K expected to
        be much smaller).
```

```
RTIME=TLE@AE@(1+(N+1)@PI@NDIS@ANDS).
```

```
LTIMEH=TLE@PU@N@NXI@AE
        @(PB1@(PI@PTRACE+PU)+PB2
        +PB3@NXI+M+(TLE MODULO PS)).
```

IX. Model Simplification

If one were to consider all of the variables in each of the above models in a simulation, the number of possible combinations would generate more results than could be readily analyzed, thus restricting the conclusions which might be made. Thus, it is convenient to simplify the models. Because the value of AI is normally quite small in comparison to the value of AE, any variables which do not multiply an AE do not contribute enough to the total processing time to be considered. These parameters were thus assumed to be zero.

The remaining parameters are in a product with AE. Using the results of previous work (7), considering the assumptions which affect the processing algorithms, and for purposes of comparison, only the parameters LE, PI and NDIS are varied in the models. The remaining parameters are assigned values which are either "best case" choices or which are simply chosen for purposes of comparison.

The best case choices involve the parameters:

ADI = 10	PB1 = .3	MEM = 10480
POC = .8	PB2 = .4	
PDP = .2	PB3 = .3	
PADV = .2	PS = 2096	

For purposes of comparison the following parameters are assigned the values shown:

N = 3
NXI = 3
NDI = 4

The parameters to be varied affect the algorithms in different ways. The parameter LE (the number of logical entries in the file) is a scalar affecting both algorithms. Its value, however, influences the amount of memory required for processing. Since the value of TLE is dependent on LE, but will normally be less than LE, less memory is required for M2 processing. Thus, there is a point at which the value of LE will adversely affect the processing time of M1, in comparison to that of M2.

The parameter NDIS (the average number of elements satisfying a given query for a logical entry) is varied because of its relation to NDI. The quotient NDIS/NDI can be considered to be the probability of an element satisfying an operand of a given Boolean query. This means that if NDIS/NDI = 1, then every element involved in the query satisfies the query. Likewise, if NDIS/NDI = 0, then no element satisfies the query. This quotient thus has important implications as to the number of nodes in the hierarchy which will be found by the right side of the request statement.

The parameter PI (the proportion of ANDs in the Boolean query, PU = 1-PI) is important because it is critical in determining the possible configurations of the Boolean query. In addition, in prior work (7) it was determined that the value of PI was very influential in the processing time for M2.

There are two independent parameters which have not been discussed. These are AI and AE. These parameters refer, respectively, to internal memory and external memory accessing time. For purposes of comparing the processing algorithms the value chosen for AI was zero and for AE was twenty milliseconds.

The simplified models were simulated. The results of the simulations and the conclusions which were drawn are presented below.

X. Simulation Results and Conclusions

The processing algorithms for the two file structures were simulated by

approximating the models of the algorithms under the assumptions presented below. The values of the parameters used in the simulation were:

LE - 1000, 9000, 25000, 45000, 65000, 85000
PI - 0.0, 0.2, 0.4, 0.6, 0.8, 1.0
NDIS/NDI-10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1.0

The results provided by the simulations were different from those previously obtained (7). These differences resulted from the assumptions presented earlier.

For retrieval, the processing time for M2 was less than that for M1 in those cases where either NDIS/NDI was less than 0.001 and PI=1, or where NDIS/NDI was greater than 0.2. When NDIS/NDI is very small and PI=1, the processing times for M1 and M2 are small and differ only slightly. In addition, PI=1 means that there are only AND operators in the Boolean query. This construction of a Boolean query would not be normal. Thus, each of the above cases does not provide realistic results. For those values of NDIS/NDI and PI which are in the range normally expected, PI<1.0 and NDI/NDIS<.1, the retrieval time for M1 is better than that for M2.

For maintenance, there are points of transition which designate when the processing time for M1 becomes and remains greater than that for M2. These transition points are given in terms of NDIS/NDI in Table 2.

From the results presented in Table 2, it is apparent that for LE $\geq$ 9000 the value of NDIS/NDI at the point of transition decreases as the proportion of ANDs increases. This is an expected result, because as the proportion of ANDs increases the number of logical entries which are processed sequentially in M2 decreases while the number of nodes to be processed in M1 remains the same.

For LE = 1000, the results are not conclusive. The values of NDIS/NDI are large because there are not a sufficient number of values of elements to entail extra external accesses while processing within the inverted file. In addition, this number of logical entries is much smaller than would normally be considered for use with these file structures.

As the number of logical entries increases, the point of transition occurs for increasing values of NDIS/NDI. These values remain small, being less than or equal to five per one thousand, for up to 85000 logical entries. In previous work (7) the point of transition was 0.1. This difference can be explained by the new assumption that a partition of unique values of an element is accessed from external memory rather than all unique values for the element.

One assumption which adversely affects the processing time for M2 is that the preprocess time for M2 is equal to the right-side processing time for M1. This is unreasonable since normally there would be considerably fewer elements in the inverted file structure of S2 than in that of S1. This is a "best case" assumption for M1.

Considering the results, the choice between S1 and S2 is not conclusive. However, S2 is more promising than it appeared in previous work (7).

In this paper all possible operations on the data structure were not considered. In previous work (7) maintenance operations involving subtrees were considered. The reason additional maintenance operations were not considered herein is that this additional detail would have been of questionable value in illustrating the methodology presented.

TABLE 2

Processing Time Transition Points*

Number of Logical Entries	Proportion of ANDs in the Boolean Query 0.0	0.2	0.4	0.6	0.8	1.0
1000	.00750	.00500	.00500	.00250	.00250	.00010
9000	.00075	.00050	.00050	.00025	.00025	.00010
25000	.00250	.00100	.00075	.00075	.00050	.00010
45000	.00250	.00250	.00100	.00075	.00050	.00010
65000	.00250	.00250	.00250	.00100	.00075	.00010
85000	.00500	.00500	.00500	.00250	.00100	.00010

*Each element of the table is the value of the quotient NDIS/NDI.

The fact that no conclusive results were obtained does not decrease the value of this paper, the purpose of which is to illustrate a methodology. This methodology is useful for comparing file structures and associated processing algorithms, and it is practical because of its simplicity. There is no requirement for a detailed simulation of all the processes associated with data management systems. This methodology is also presented as an approach which should be considered for solving many problems which defy solution due to their complexity. For the particular problem considered herein, choosing a file structure, no definitive preference is established between two alternatives, given the stated set of assumptions. But this information is significant in a system design environment in considering the desirability of implementing an alternative storage structure in a given system.

REFERENCES

(1) EVERETT, G. D. Data structure and algorithmic determinacy, unpublished Ph.D. dissertation. The University of Texas at Austin, May, 1971.

(2) DALE, A.G.; EICHELKRAUT, MIRIAM. General information, remote file management system, phase I, TSD-0. The University of Texas at Austin Computation Center, February, 1968.

(3) EICHELKRAUT, MIRIAM. Systems programming documentation, TABLES, remote file management system, phase I, TSD-1. The University of Texas at Austin Computation Center, October, 1968.

(4) LEFKOVITZ, DAVID. File structures for on-line systems, Washington, D.C., Spartan Books, 1969.

(5) CHAPIN, NED. A comparison of file organization techniques. Proceedings of the 24th National Conference, ACM, 1969, p. 273-283.

(6) MARTIN, LAURENCE D. A model for file structure determination for large on-line data files, File Organization, Selected Papers from File 68--An I.A.G. Conference, Swets and Zeiglinger N.V., Amsterdam, 1969, p. 223-245.

(7) WINKLER, ANTHONY J. A methodology for comparison of file organization and processing procedures for hierarchical storage structures, unpublished Ph.D. dissertation. The University of Texas at Austin, May, 1970.

(8) FLORES, IVAN. Analysis of internal computer sorting, JACM 8, 1961, p. 41-80.

AN APPROACH TO RESEARCH IN FILE ORGANIZATION

Daniel Teichroew
The University of Michigan

ABSTRACT

Research in file organization is a problem in which the unstructured efforts of many individual researchers have not produced results commensurate with the effort expended. The paper briefly examines some of the reasons for this and suggests a structure consisting of a language and terminology for communication, a model of information processing systems and tools for studying and analyzing problems related to such systems. The ISDOS project is outlined as a coordinated approach which may serve as the beginning in the development of a satisfactory structure.

KEY WORDS AND PHRASES

file organization, research methodology, information processing systems

1. Introduction

A considerable amount of effort has been expended in the past fifteen years in studying the file organization problem. Some progress has been made, particularly in the practical area--various file organizations are implemented in a number of software packages. From the point of view of a theory of file organization, sufficiently powerful to help in making the decision of which file organization should be used in a given situation, the progress, however, has not been commensurate with the effort expended.

The amount of resources in the economy spent on computer-based information systems is substantial and the effectiveness of a substantial part of this depends on how well the files are organized. It is, therefore, of considerable importance to determine why this subject has not progressed more rapidly, and, even more important, to determine what can be done to improve the effectiveness of research effort in the future.

File organization is a very complex, large subject with many ramifications. It is a basic thesis of this paper that a major reason why more progress has not been made is that researchers, or research projects, all have a tendency to attack the whole problem. Since each one has limited resources, only a small dent in the total problem can be made and it is frequently equivalent to that already made by others. Consequently, rather than having the result of n man-years of effort, we have the result of, say, one man-year of effort, expended n times.

The reason researchers feel a necessity to take a broad comprehensive view is that there exists no framework into which the solutions of "micro" problems can fit and still make a contribution to the total. The researcher working on the micro problem cannot see the value of his contribution, nor for that matter can the profession, because it does not contribute in a recognizable way to progress in understanding the whole subject.

The situation is somewhat analogous to an attempt to build a bridge by assigning, say 1000 engineers, to designing the bridge. Since only one design can be used, the effort of the other 999 is mostly wasted. The only way that the task can be accomplished efficiently is to first have one, or few engineers, develop an overall design. (Frequently, of course, it is worthwhile having several competing designs produced so that the final design chosen can incorporate some desirable features of the others.) Once the overall design is chosen the other engineers can fill in the details of their parts, secure in the knowledge that this work will make a contribution because it is part of a total picture.

Like all analogies this one is imperfect, but it does point out the need for some way of developing the overall picture to which individual efforts can contribute. Certainly, developing a comprehensive model of file organization more closely resembles the building of a bridge than it does the growth of a coral reef by accretion of many individuals in an unorganized, unplanned way.

If it is granted that a structure is needed, the next step is to determine an effective one for the study and development of the file organization. The structure proposed in this paper consists of three elements: (i) a language and associated notation and terminology, (ii) a model of information processing systems of which file organization forms an integral part and (iii) tools for studying and analyzing the problem.

(i) Language and Terminology

Since a relatively large number of individuals are, and will be, working on the file organization, it is absolutely necessary that they have an effective method for communication. Furthermore, there is an even larger group interested in applying results and reporting back on performance and effectiveness. The basic element of the communication medium must be terminology. The only effective method for establishing and maintaining

a living terminology is by a discipline, enforced by referees and editors, that authors explicitly define the terms they use. Where their definitions differ from accepted ones it is the author's responsibility to indicate how, and more importantly, why his differs. Unfortunately, this situation does not yet exist in file organization. Even the very basic term such as "file organization," "record," "storage structures" are used without definition by various authors with widely different meanings.

A standard terminology cannot be maintained unless it is enforced in some way. It is not likely that referees and editors will be able to enforce a system even if they want to, since so much is published outside the normal (refereed) system. A better hope for this is a formal analysis tool which can only be used if standard terminology is used.

(ii) Model of an Information Processing System

As emphasized earlier, file organization is in itself a large complex problem. However, to study it effectively, it must be placed in context of the even larger problem of which it is a part. The basic reason for this is that without a broader context the problem of choosing a "best" file organization is indeterminate--there are many parameters and too many possible definitions of "best." In any particular case, the values of many of the parameters and the specific definition of the objective function is fixed by the broader context. This is just another way of stating that files do not exist solely for themselves--they exist as part of a system.

The basic element of the model is an information processing system which can be represented as a black box as shown in Figure 1. The black box exists to produce outputs and to accept inputs.

The inside of the black box, shown in Figure 2, contains noncomputerized procedures, procedures for man-machine interfaces, and an inner black box, the "physical system," which contains hardware, hard software, programs and files. The task of constructing the physical system then consists of selecting the hardware and hard software, and building the programs and files.

The next step in developing the model might be to identify and characterize "basic" types of systems with the criterion being related to how the files interact with the rest of the system. For example, three different types of systems are:

> Information Storage and Retrieval - data is added to files without a knowledge of the number and type of enquiries that will be made.
>
> Command and Control - data is added to the files, again without the use being specifically known; however, the data is used by a monitoring program which looks for certain conditions occurring.
>
> Business Data Processing - the data stored in files is there as a result of a specific known need.

(iii) Tools for Studying and Analyzing the Problem

The need for a langugage for business analogous to what mathematics is to science is often stated, e.g., by Vaughn (13):

> "When a scientific problem is presented to an analyst, the mathematical statement of the relationships that exist between the data elements of the system is part and parcel of the problem statement. This statement of element relationships is frequently absent in the statement of a business problem. The seeming lack of mathematical rigor has led us to assume that the job of the business system designer is less complex than that of the scientific designer. Quite the contrary--the job of the business system designer is often rendered impossible because the heart of the problem statement is missing!
>
> The fact that the relationships between some of the data elements of a business problem cannot be stated in conventional mathematical notation does not imply that the relationships are any less important or rigorous than the more familiar mathematical ones. These relationships form a cornerstone of any system analysis, and the development of a problem statement notation for business problems, similar to mathematical notation, could be of tremendous value."

Mathematics is a language for humans--humans can, after some study, learn to comprehend it and to use it. It is not at all clear that the equivalent language for information systems problems will be manipulatable by humans though obviously it must be understandable to them. Computer manipulation is necessary because the problem is so large and a person can only look at one part at a time. The number of parameters is too large and their inter-relationship is too complex. There may be a hope that the number of parameters that are important will be reduced if a better model is developed. In the way, say, that the number of significant parameters was drastically reduced when ellipses were substituted for circles in the model of the motion of the planets in the middle ages.

These statements apply to the type of information processing systems in which file organization are important. Consequently, what is needed is not just the notation and syntax referred to in (i) but also the software package that will manipulate a problem statement for human use.

FIGURE 1. EXTERNAL VIEW OF AN INFORMATION PROCESSING SYSTEM

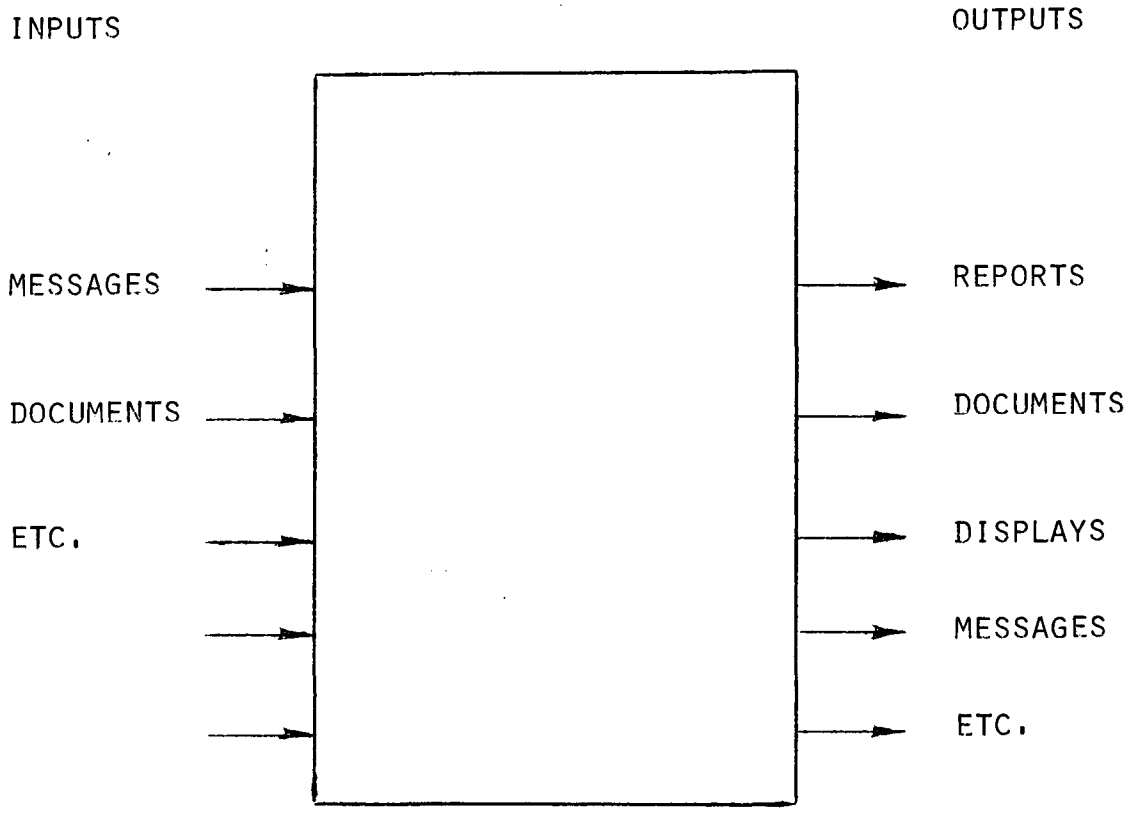
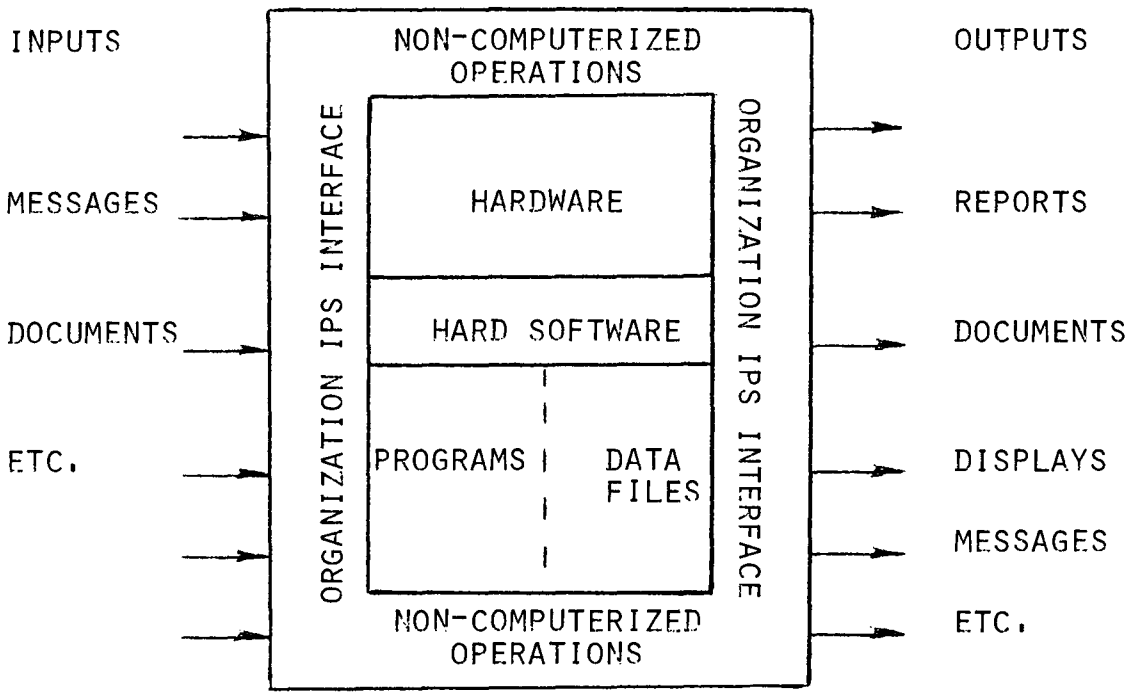


FIGURE 2. INTERNAL VIEW OF AN INFORMATION PROCESSING SYSTEM



AN INFORMATION PROCESSING SYSTEM (IPS) IS A SET OF HARDWARE, SOFTWARE PACKAGES (HARD SOFTWARE), COMPUTER PROGRAMS, PROCEDURES AND PERSONNEL THAT HAVE BEEN ASSEMBLED AND STRUCTURED SO THAT THE WHOLE SET ACCOMPLISHES SOME GIVEN DATA PROCESSING REQUIREMENTS IN ACCORDANCE WITH SOME GIVEN PERFORMANCE CRITERION.

2. A Proposed Structure

The ISDOS (Information Systems and Optimization System) project represents one possible structure for studying file organizations that satisfies the specifications described here. The ISDOS project is basically concerned with developing methods for designing and constructing Information Processing Systems--as automatically as possible. It is not possible to here describe ISDOS (a preliminary description is given by Teichroew (10)); but an outline will be given to indicate how the specification can be accomplished.

The reason why the ISDOS project may be a satisfactory framework for studying file organization is a result of a number of premises on which the project is based. The ones that are related to the file organization problem are as follows:

(i) There is no one file organization which is always "best" under all circumstances. Which file organization is "best" depends on a number of factors.

(ii) It follows from this, then, that what one needs is a way of selecting the appropriate file organization on the basis of the available information at an appropriate point and time. Then, since changing file organization may be expensive and time consuming, the stages in system construction which follow the choice of a file organization, and which are easier to modify, should then be selected optimally given the particular file organization that has been chosen.

(iii) Despite its potential cost there must be some way of changing file organization periodically because there arise situations, due to changes in requirements, hardware or methodology, when changing the file organization is less expensive than maintaining the old system. To reduce the cost of such change the method should be as automatic as possible.

(iv) What makes the selection of a "best" file organization difficult is that there are a large number of possibly conflicting requirements that the system (and the file organization) may have to accommodate. It is therefore essential to look at the various types of requirements, arising from the fact that systems of the future will be a mixture of the basic types mentioned above.

(v) The "best" file organization cannot be obtained without being able to describe what is known about the needs of the system.

(vi) The method of selecting files must be part of, or related to, the method of deciding the best processing sequence since there is usually a tradeoff between memory capacity and computing time.

(vii) Because changes in requirements occur it is necessary to be able to monitor the performance of systems after they are in operation. If inefficiencies are detected, it is desirable to be able to make modifications to improve effectiveness.

To date, the building of the systems is accomplished primarily by manual methods with relatively little assistance from computer aids. The major exception to this is that the computer is used in the construction phase to translate source language programs into object code. The selection of a file organization is usually made by a number of analysts or programmers within the constraints of the software supported by manufacturers.

The ISDOS project begins with the idea that there should be a clearer separation between what the system is required to do and the method by which it is accomplished using the hardware, programs and files. In order to do this it is necessary to have a formal way of stating the requirements which can be investigated analytically. Since the requirements are large, it is only practical to do this analysis using computer programs. Available methods of documentation using first the English language supplemented by flowcharts and then general purpose programming languages do not permit computer manipulation of requirements.

The ISDOS system is based on a problem statement language which does require statement of problems in a way that does not limit automatic manipulation of these requirements. This statement of requirements is then processed by a series of algorithms, currently being implemented in computer programs, which will permit the automatic design and construction of the systems. The language and the algorithms will be described briefly.

(i) Problem Statement Language

The Problem Statement Language is a language people use to write their requirements that the target Information Processing System is to satisfy. The Problem Statement Language is the crucial part of ISDOS since it forms the interface between the organization and ISDOS. It must, on the one hand be suitable for use by the Problem Definer, and on the other hand provide information in sufficient detail for the succeeding parts of ISDOS.

General purpose programming languages such as COBOL, FORTRAN and PL/1, are not satisfactory for use as requirement statement languages though they are similar in that they must both have sufficient structure to be analyzable by computer programs.

The requirement statement language being developed in the ISDOS project is known as PSL (Problem Statement Language). PSL can be considered a generalization of languages such as those of Young and Kent (12); Information Algebra (CODASYL

Committee (1)); SYSTEMATICS (Grindley (2), (3)); TAG (IBM (5)); and ADS (National Cash Register (9)).

A problem statement in PSL consists of a number of statements grouped into units called problem statement units (PSU). Each PSU describes an input or an output and consists of four major parts:

- An identification section which contains statements relating the input and output to the world external to the IPS.
- A Time-and-Volume section which describes the timing of the input or output and parameters which determine the volume.
- A Data Structure definition section which describes the structure of the data.
- A computational definition section which states the formulas defining the variables which must be calculated.

Another part of PSL is used to describe requirements for the system as a whole: system-wide parameters, constraints, capabilities of hardware and software available and the performance criterion.

(ii) Problem Statement Analyzer

The statements made in PSL must be analyzed to make sure that the requirements are consistent and correctly stated. The Problem Statement Analyzer (PSA) performs this function in ISDOS. It is a software package that accepts problem statements in PSL, and maintains files necessary for its use, carries out analyses and produces output. The analysis is carried as far as is possible without bringing in hardware considerations. The process of statement analysis is monitored by a System Definer and a Data Administrator who in turn communicate with the Problem Definers and their managers. When the total problem statement has been analyzed, the output of PSA is made available to the Systems Optimization Design Algorithm (SODA) and the Code Generator (CG). Some requirements for information of the type produced by PSA have been discussed by King (6) and in the TAG manual, IBM (5). The Problem Statement Analyzer is the tool which amplifies the capability of an individual to understand and comprehend the requirements that have been stated.

(iii) System Optimization and Design Algorithm

SODA, is a software package which accepts the results of analysis of problem statements by PSA and develops, for a complete problem statement, a specification for the target system by selecting from alternatives available, subject to stated performance criteria. It develops specifications for programs and files by searching the available alternatives for the optimal solution according to constraints of time and volume, hardware alternatives (CPU speed, memory size, number and characteristics of peripheral units, etc.) and costs. This algorithm continues in the path of formalizing systems design techniques pioneered by Langefors (7), Grosz (4), Turnburke (11), Martin (8), etc. SODA produces either (a) diagnostics which may help the system definer and problem definers restructure or modify the requirements or (b) a set of specifications. Again, in practice several people interact with SODA--notably, the System Definer, the System Designer, the Hardware Acquisition group, and the Performance Officer.

Should the specifications produced by SODA be approved, details of the designs produced by SODA are passed on to the Code Generator (CG), the Data Re-organizer (DRO) and the System Director (SD) for actual construction.

(iv) Data Re-Organizer

This is a software package that accepts specifications for storage structures from SODA. In addition, it also takes existing data stored on some physical device along with its structure and re-organizes the data on the new hardware in accordance with the specification produced by SODA and produces a storage structure directory for use by the System Director.

(v) Code Generator

The Code Generator is a software package which accepts the specification produced by SODA (and possibly detailed problem statement information from PSA and/or detailed specifications from DRO) and produces an output which the System Director will accept in order to produce executable machine instructions or parameters for generating them. This output may be in the form of object code, parameters for software packages or a higher level language to be interpreted.

(vi) System Director

System Director (SD) takes the output from the CG and the storage structure directory from the DRO and produces the target IPS. The IPS designed and constructed by ISDOS will consist of hardware (selected or approved by ISDOS), software (designed by ISDOS) and files (structured by ISDOS). The IPS will then be ready to deal with routine operational input from the external environment (e.g., customer orders, etc.) and will produce output (e.g., invoices, etc.). This will cause changes to the contents of its files (e.g., changes to the quantity on hand, etc.). In addition, SD is capable of monitoring system performance. This last facility would be used to aid the Performance Officer and ISDOS in maintaining an efficient system.

3. Discussion of the Proposed Structure

The specifications of the ISDOS approach and the organization described above were

developed to achieve the benefits of automation in the systems building process: shorter elapsed time between statements of requirements, ability to accommodate changes at the requirements level automatically, ability to state requirements independent of hardware considerations, etc.

It can become a structure for the study of file organization problems (among others) as a corollary to the fact that it is based on the use of the computer to aid in statement of requirements, analysis, design and construction.

It satisfies at least part of the need for a standard language and terminology. Since the requirements stated in the problem statement language are processed by computer programs, it is these programs which enforce the standardization at a much higher level than can be achieved with manual methods. While complete standardization enforcement by computer programs may never be possible at least a large part of the terminology can become standard.

It is based on a standard model of an information processing system. The use of the Problem Statement Language and Problem Statement Analyzer and auxiliary function such as Data Administration will make it possible to continually refine the model of Information Processing Systems and facilitate the identification of common features for various systems. Consequently, the methodology being developed will have much broader application.

It will provide a consolidated view of the system. This will make it possible for management to have control over the system from an overall point of view. A particular manifestation would be the organization of files in a manner best suited to maximize the organizations objective function, satisfying all defined constraints, rather than a biased localized file structure enforced ad hoc by the group with the most current problem.

The tools for studying and analyzing problems are provided because all statements of requirements are processed by computer programs, particularly the Problem Statement Analyzer. The programs become the tool that can carry out analysis of problems that are too large and complicated for any one individual or even a group to carry out. The development of the analysis, of course, must be done by humans, but the application can be aided by the use of the computer as a result of the Problem Statement Language and Problem Statement Analyzer.

The requirements for the algorithm for selection of file organization will be expressed in the Problem Statement Language, (as will the rest of the ISDOS algorithms). Consequently, these can be analyzed using the computer as a tool under the control of the Problem Statement Analyzer.

4. Conclusion

The ISDOS approach represents one framework in which research and development in file organization can be structured to yield a cumulative effect. It is based on the concept that the problem of file organization, and the problem of information processing systems in general, is so large and complex that progress can only be made by utilizing the computer itself as an integral tool in the process. The approach is pragmatic in that brute force methods, if necessary, are used initially to produce results. Refinements and elegance can come later.

REFERENCES

(1) CODASYL Development Committee. An information algebra-phase I report. Communications ACM, 5:4 (April 1962) p. 190-204.

(2) GRINDLEY, C.B.B. Specification and interrogation of formal control systems. Paper presented at TIMS XVII Conference, London, July 1-3, 1970.

(3) GRINDLEY, C.B.B.; STEVENS, W.G.R. Principles of the identification of information. File Organization. IAG Occasional Publication No. 3. Scolts and Zeitlanger N. V., Amsterdam, 1969, p. 60-68.

(4) GROSZ, M.H. Systems generation output decomposition method. Standard Oil Company of New Jersey, July 1963.

(5) IBM. The time automated grid system (TAG): sales and systems guide. Publication No. Y20-0358-D, 1968.

(6) KING, P.J.H. Systems analysis documentation: computer aided data directory definition. Computer Journal, 12:1 (February 1969).

(7) LANGEFORS, B. Theoretical analysis of information systems. 2 Vol. Studentlitteratur, Lund 1966. (Also available from National Computing Centre Ltd., Quay House, Quay Street, Manchester, England.)

(8) MARTIN, J. Design of real-time computer systems. Prentice Hall, Englewood Cliffs, New Jersey, 1967.

(9) NATIONAL CASH REGISTER COMPANY, Accurately defined systems. 1967.

(10) TEICHROEW, D. Methodology for the design of information processing systems. In Proceedings of Fourth Australian Comp. Cong., August 1969.

(11) TURNBURKE, V.P., Jr. Sequential data processing design. IBM Systems Journal (March 1963).

REFERENCES (continued)

(12) YOUNG, J.W.; KENT, H. Abstract formulation of data processing problems. J. of Ind. Engr. (November-December 1958) pp. 471-479. Reprinted in Ideas for Management, International Systems-Procedures Association., 1959.

(13) VAUGHN, P.H. Can COBOL cope. Datamation (September 1, 1970) pp. 42-46.

QUANTITATIVE EVALUATION OF DESIGN TRADEOFFS IN FILE SYSTEMS

C. P. Wang and V. Y. Lum
Information Sciences Department
IBM Research Laboratory, San Jose, California

ABSTRACT:

The design of a file system has never been a simple nor a straightforward task because of its complexity. Heuristics and experience still play a major role in guiding the design process. To organize the entire design process in a more systematic manner, large scale simulation has proved to be an effective technique. The FOREM models developed during the past several years (specifically for the evaluation of file system designs) represent facilities of this type. This paper utilizes the FOREM model as the principal tool and presents a hypothetical design example dealing with many essential issues of the design process. Evaluation of designs of several other actual file systems are also being carried out and will be reported at a later date. Only through quantitative evaluation can each design decision be arrived at correctly and the possible tradeoffs be identified.

KEY WORDS AND PHRASES

file design, file organization, file simulation and modeling, information retrieval, secondary indexes, data management, secondary keys, inverted file, quantitative design evaluation

I. INTRODUCTION

The complexity of an information system is manifested in the form of at least three major levels of system abstraction each having a set of assorted primitive concepts, design parameters and issues. At the user's problem or logical level, the application analyst deals with sets of objects or entities, their properties and relationships. He may choose a set of definition schemata to declare entities and the pertinent relationships and provide procedures for their manipulation. In this environment, the issues to be dealt with are the user's requirements, his desired output, and the application algorithm. At the access path design level of the systems, the choice of data structure for the data base depends on the combined requirements of all the existing and prospective users of this data base and the projected transaction loads. Here, the data elements may be combined in a logical manner different from that specified by the user's schemata to facilitate search and access. Other facilities such as indexes and cross reference tables are also introduced here to improve the efficiency of finding the appropriate item in the data structure. The data base designer has to provide the proper search algorithms for transforming the users specification of any subset of entities into search steps which are compatible with the data structure of the data base. At the physical level, the chosen structure of the data base must be mapped into the addressing space of the storage device in the host computer and the idiosyncrasies of the data management software have to be taken into account before making the design decisions.

Specific tasks at each level of the design process involve respectively the characterization of the application, the selection of an appropriate file organization, and the determination of many hardware and software parameters to implement that organization. Since the types of applications supported by a file and the resulting transactions are almost unlimited in variety, no single design is equally good for every application. The problem of file design is essentially that of tailoring the file organization to the application in such a manner that some objective function of the system resource requirement, and the aggregated processing time is a minimum. In case the file system provides on-line inquiry capability from remote terminals, additional constraints such as the user's specified response time requirement should also be incorporated in the design.

Many analytical studies have been done in the past on the problem of file design, but most of them concentrated on the filing schemes and the associated access techniques (1,2). More recently, some work has been initiated to treat the file system as a comprehensive problem of design based partially on known optimization techniques (3,4,5). Because of the large number of variables associated with a file system, success is rather limited and is largely confined to files with the simple sequential organization (6,7). Design and optimization of the file system in the general sense seems to remain a distant possibility. At present, the most effective means in carrying out the file system design in a systematic way is through large scale simulation. In this connection, the FOREM (File Organization and Evaluation Model) model has been developed to facilitate the evaluation of various file organizations.

The FOREM model, through the use of many parameters, allows the modeler to specify all three levels of abstraction with sufficient complexity. Extensive use has been made of the model in evaluating many design alternatives at the access path--physical levels, including the organization of data sets, the selection of access methods and many subtle issues such as the blocking factor, the overflow handling technique, the buffering scheme (8), etc. In this paper, we shall present a hypothetical yet quite realistic example of file and application as a vehicle to illustrate the logical sequence of the design process and the associated

issues. The FOREM model provides a convenient probe for the quantitative verification of these issues, their alternatives and possible tradeoffs. It is the authors' intention to accumulate sufficient insight and experience through this method of quantitative evaluation to guide the development of a systematic methodology of file system design and optimization.

II. THE FOREM MODEL

Much of the detailed description on the functional as well as operational aspects of the FOREM model has been discussed elsewhere (9,10). To retain continuity without unnecessary duplication, we shall describe only those features of the model that are particularly relevant for the design issues discussed in this paper.

The FOREM model is a programming facility allowing the modeler to simulate file management systems and the associated dynamic storage and retrieval activities. The facility calculates the total allocated storage space and computes the processing time for each transaction. In its present form it can accommodate more than one hundred types of variables which we shall not attempt to list here. However, it should be beneficial to review at least some essential ones in the framework of the above-mentioned three levels of system abstraction.

The model does provide a rudimentary facility for the specification of user's data elements and their relationships. The elementary unit of data is a field. A contiguous set of fields is called a segment. Different segments can be organized hierarchically to represent complex relationships among data elements. In addition, user's retrieval, update and insertion operations can also be expressed in tabular form with qualification on keys as well as on other field values. In a qualification expression, the allowable predicates include equal, greater than, less than and range. The specification at users level is essentially independent of the access path or physical levels of the file. For the specification of the data base, the notion of access-path-level data sets is introduced. All segments and their relationships specified in the user's schemata can be mapped into a single or several data sets at the discretion of the designer. Facilities are also available for the specification of indexes and cross reference tables which are not part of the user's schemata. Basic mechanisms for the construction of search strategies in tabular form are also provided to transform the user's storage and retrieval specifications into search steps directed to the appropriate data sets.

To enable the users to design the storage implementation of the access-path-level data sets, the model is capable of simulating a number of storage devices such as tape, drum, data cell and disk, and several standard access techniques supported by the IBM 360 operating system. These include sequential, indexed sequential and direct access methods. Other storage and access technique related parameters such as blocking factor, overflow handling technique, buffering mode, etc. are also provided as design variables.

As outputs, the model calculates the storage space occupied by each data set and the processing time (elapsed time) required for each user transaction. The designer can then use these two outputs as basic indicators to guide his entire design process.

III. APPLICATION DESCRIPTION

The best approach to provide an illuminating insight to the entire design process and the attainable tradeoffs is to follow through the design steps of a file system accompanied by the rationale behind each step. The first step to start that process is to provide a parameterized description of the application and the associated collection of data items. Consider a hypothetical application dealing with three sets of objects called VENDORS, PARTS, and PROJECTS. Each vendor is characterized by a unique identification NUMBER, a NAME, and an ADDRESS. Each part has the attributes PART NUMBER and DESCRIPTION while each project has the attributes PROJECT NUMBER and PROJECT NAME. In addition, the application is also dealing with the 3-tuple relationship (VENDOR NAME, PART NUMBER, COLOR) and the 4-tuple relationship (VENDOR NUMBER, PART NUMBER, PROJECT NUMBER, QUANTITY). The user's declaration can conceivably be given as

FIELD DECLARATION

LEVEL	FIELD NAME	ENCODED CHARACTER LENGTH
0	VENDOR NUMBER (KEY)	6
0	NAME	10
0	ADDRESS	60
1	PART NUMBER	8
1	COLOR	8
1	DESCRIPTION	20
2	PROJECT NUMBER	8
2	QUANTITY	4
2	PROJECT NAME	16

SEGMENT DECLARATION

SEGMENT NAME	FIELDS IN SEGMENT	SUPERIOR SEGMENT NAME	AV. NO. OF REPETITIONS
S0	VENDOR NUMBER	--	1
	NAME		
	ADDRESS		
S1	PART NUMBER	S0	2
	COLOR		
	DESCRIPTION		
S2	PROJECT NUMBER	S1	3
	QUANTITY		
	PROJECT NAME		

One occurrence of such a record as viewed by the user is shown in Fig. 1. It is interesting to note here that there exist many-to-many relationships among the entities called VENDOR, PART, and PROJECT and the above specification is certainly not the only one possible. The question of what

are the most appropriate schemata for a given application is an interesting area of investigation beyond the scope of this paper. Here, we simply postulate that the particular choice of such a declaration is a result of a careful analysis of the application. To preserve the simplicity of the problem, let us further assume that it is the only application supported by the file management system.

In addition to the segment declaration given above, information about the field values, the inquiry and maintenance activities, and the corresponding occurrence frequencies are also necessary to provide a complete characterization of the file and the associated applications. For those fields used in the selection criteria of user's storage and retrieval requests, their value sets and occurrences of each value in the file enable a more accurate determination of the total I/O activities to be initiated in connection with each user's transaction and, hence, a more accurate estimate of the processing time of that transaction.

The dynamic inquiry and update activities generated by the user's application of this file are represented by the set of hypothetical queries given in Table I. Although it appears that the query set is made up in an arbitrary manner, it can be divided into four broad classes, namely (i) qualification on one or more fields in the master segment and output either the entire record or segments at any subordinate level, (ii) joint qualification on fields in both the master segment and the subordinate segment and output either the entire record or any segment, (iii) qualification on fields in one or more levels of subordinate segments and output the entire record or the master segment, and (iv) given record key update the entire record or some fields in the record. It should be noted that the frequency of occurrence associated with each inquiry and update operation has the significance of weighting function in their effect on the outcome of the design. No record insertion operation is given here. It is assumed that the file is periodically reorganized in a batch maintenance mode when sufficient number of new records are accumulated for insertion.

It can be observed from Table I that only four fields, "VENDOR NAME," "PART NO.," "COLOR" and "PROJECT NO." are used in the selection criteria of any user's transaction. To generate the number of records retrieved from the qualification criterion of a query, the following two sets of hypothetical field value distributions are provided in Table II. Although these two sets of distributions differ only in the field "COLOR," it should be interesting to observe their effect on the result of the design.

The application description is a crucial aspect of the design process. Great care should be exercised to ensure that the description is completely independent of the access path and physical implementations of the files. The types of characteristics given above provide a framework for extracting appropriate parameters.

IV. EXAMPLES OF DESIGN

The first major design issue to be resolved is the selection of a logical organization for the data base. In the general case, a data base may support a number of applications and conceivably has an organization quite different from that specified by the user's schemata. The problem of selecting the best schemata for the data base taking into account the requirements and activities of all supported applications is an extremely difficult and unresolved problem. It is certainly the key to a systematic approach of data base system design and optimization. In the hypothetical example considered in this paper, the situation is much simpler because there is only one application associated with the data base. We shall explore two access-path-level organizations in more detail in the following:

A. Choice of Data Base Organization

The first access-path-level organization of the data base to be investigated has the same format as that declared by the user. Assuming that each encoded character of every field takes one byte (8 bits) of space, the length of the record can then be readily calculated. We shall also postulate that there are 10^5 records in the data base. This set of records can be sorted in collating sequence of the keys into an access-path-level data set. It should be noted that the sorting order introduced in the data base may be quite different from that envisioned by the user.

Next, let us consider the data set so specified and the search technique required in finding records in the file which satisfy the selection criterion in any given query. Of course, the set of records satisfying a given query listed in Table I can be found by scanning the entire file sequentially in a record by record manner. But for fast retrieval, via key, an index table of record keys can be built. It is generally recognized that cross reference indexes organized into separate data sets can also be created on those fields which are used in the qualification criterion of any query. From the query Table I, it is clear that four fields "VENDOR NAME," "PART NO.," "COLOR," and "PROJECT NO." are potential candidates for the creation of cross reference indexes. However, extra maintenance effort is required to add a value or a key entry or both to each of the cross reference indexes during the insertion of a new record to the file. When updating the field value of a record, extra work in modifying the cross reference table may be necessary if that field is indexed. In any event, the merging time of two long key lists obtained from two different cross reference indexes for joint qualification on two or more fields may be quite substantial. Therefore, the design problem is to balance the benefits of reduced retrieval times against the drawbacks of extra index processing and maintenance efforts.

Consider now the processing time for the query Q10 in Table I. Suppose all four fields mentioned above are indexed. Then there are three

possible search strategies to find all the qualifying records. The first strategy starts from the cross reference indexes on both fields "PART NO." and "COLOR", retrives one qualifying record key list from each and then performs the intersection operation. The end result is a shorter key list which can subsequently be used for record retrieval. The second strategy retrieves only the record key list from the index on the field "PART NO." and then uses that for the selection of record. The third search strategy, of course, uses only the cross reference index on the field "COLOR." Since the first strategy in general produces the shortest key list for record retrieval, it is usually but not always the best strategy in most situations. However, for the hypothetical file under consideration, the field "COLOR" has only 4 distinct values. The length of the key list associated with any value entry in the cross reference index on "COLOR" is at least 10^4 key entries or more. Consequently, a great deal of time may be consumed in key list accessing and merging. All three cases of cross reference indexing will be evaluated quantitatively.

B. Selection of Physical Structure

The next stage of design deals primarily with the physical structures and their implementations. The choice is limited, of course, to the available options of the software and hardware facilities provided by the host computer. In this example, we shall consider only the disk storage device because of the real time nature of the inquiries. The design evaluation tool, the FOREM model, is able to simulate operations of the three principal access methods, namely the sequential, the indexed sequential and the direct access methods. Qualitatively speaking, the sequential organization is best suited for batch-type sequential processing. It is not suited for inquiries with qualification on any field other than the key field. The direct organization gives rise to the shortest retrieve time under the random retrieval mode, but yields relatively poor performance under sequential processing. In other words, for any retrieval criterion based on a non-indexed field, a time consuming bucket-by-bucket search will result. When there is substantial insertion of new records into the file, the direct organization is also likely to deteriorate in performance due to the necessity of chaining the overflow records from the home bucket. The indexed sequential organization provides the capability for both sequential and random processing and can tolerate moderate amounts of insertion without severe performance degradation. However, the data set does have to be reorganized periodically when the overflow area becomes heavily populated. For the set of queries given in Table I, the index sequential organization seems to be the appropriate choice.

Besides the selection of an organization for a data set, there are several additional design parameters which must be determined. For the disk storage device, records must be blocked; the blocking factor determines the number of interblock gaps on a track and hence determines the number of records that can be stored in a track. It also determines the size of buffers that must be allocated in core. The average record length in the ISAM data set is 316 bytes and is grouped 5 records to a block on a 2314 disk storage device. Since there are 7294 bytes per track on the 2314 disk, the choice will produce four blocks to a track. The cylinder overflow mode is assumed here and there are 5% of the records in the overflow area. In addition, the four cross reference indexes are also organized into index sequential data sets with record lengths from 8 to 20 bytes, 100 records per block and 1% overflow.

After all the design parameters were selected and entered into the FOREM evaluation model, the total retrieval time for the query Q10 was computed for the three different search strategies discussed earlier. The resulting retrieval times were 52.7 sec, 21.4 sec, and 6896 sec, respectively, favoring the strategy using only indices on "VENDOR NAME," "PART NUMBER," and "PROJECT NUMBER."

It might be argued that the principal reason for the creation of an index on the field "COLOR" is to eliminate a time consuming sequential search of the complete file, when "COLOR" is the sole field used in the selection criterion of an inquiry. An inspection of the query set in Table I reveals that the appearance of "COLOR" in the selection criterion of any query is always accompanied by another field. This observation essentially leads to a conjecture that the creation of an index on "COLOR" does not serve any useful purpose. This conjecture has been borne out by the results obtained from many simulation runs. Table III gives the total time for processing the inquiries and updates in the query set for three different cases of index selections: (i) all four fields, "VENDOR NAME," "PART NO.," "COLOR," and "PROJECT NO." indexed, (ii) all except "PART NO." are indexed, and (iii) all except "COLOR" are indexed. Clearly the best performance results when no index is created on "COLOR." This simple example has demonstrated that although the creation of an index on a field is generally viewed as beneficial, the benefit cannot be ascertained without quantitative evaluation.

Now the dependence of the system performance on the content of the field can be readily demonstrated. Of course, the distribution of a field in the real world can never be arbitrarily assigned. It is an intrinsic characteristic of the file content. To demonstrate this dependence, let us consider an alternate field value distribution given in Table II. All other conditions remain the same as those in the previous design. For this alternate distribution, the total processing times for all the inquiries and updates are recomputed using the FOREM model and placed in Table IV. The results indicate that total service time of the entire query set has been reduced and that, contrary to the earlier conclusion, an implementation with indexes on all four fields now gives the best performance.

This example clearly illustrates our earlier contention that the content distribution character-

ization should be a part of the application description.

C. An Alternate Data Base Organization

It would be beneficial to examine another choice of the logical organization of the data base which does not have the identical notion of records as that envisioned by the user. For example, each hierarchical record envisioned by the user (Fig. 1) can be mapped into two different types of records of the data base with one VENDOR segment and two PART segments grouped together as a record of one type and each PROJECT segment as a record of another type. Figure 2 depicts the mapping and the resulting record types. A new key or identification number must be assigned to records of the second type. Then the records of each type are grouped together in sorted collating sequence of the keys or identification number to form two data sets. The first data set has 10^5 records and a record length of 148 bytes. The second data set has 6×10^5 records of 28 bytes per record. It may have a sorted sequence different from that shown in Fig. 2. However, to preserve linkage among segments belonging to the same original user record, pointers are imbedded in every record of the first data set to link to the appropriate records in the second data set. As far as the choice of the cross reference indexes is concerned, creation of indexes on field "VENDOR NAME," "PART NO.," "COLOR," and "PROJECT NO." seems still a reasonable design decision.

The remaining task for the designer is, of course, to choose a physical organization to place the two data sets on a disk storage device such as 2314. Here we examine only two cases. In the first case, both data sets are organized as indexed sequential data sets. The first data set has a blocking factor of 8, while the second data set has a blocking factor of 60. Both data sets are assumed to have 5% overflow records. In the second case, the first data set is organized into an indexed sequential data set while the second data set is organized as a direct access data set.

Let us now consider the case that both data sets are indexed sequential. For such an organization of the data base, the query processing time depends critically on the query type and levels of segments to be retrieved. For example, to retrieve a complete user record, the transaction must first go to the master segment of the user record in the first ISAM data set, and then follows the pointers to retrieve all the subordinate segments in the last level which is also an ISAM data set. On the average, there will be six accesses to the last level data set per user record retrieval. Table V presents the processing times for individual query as well as for the entire query set for this implementation. The results show that overall processing time will be increased substantially.

Next let us consider the case that the second data set is organized as direct access data set. All the processing times for the query set were recomputed as given in Table V. The result clearly favors the direct access organization for the second data set.

A comparison of Table V with Table III shows that for some queries Q2, Q3, Q5, Q7, Q12, Q15, and Q16, the processing times are shorter in the alternate data base organization. But for the entire set of queries, the first data base organization gives rise to shorter aggregate processing time. This may not always be the case. When both qualifications and output are confined to segments in the first two levels, the two data set organizations may become favorable. Further, there are several other organizations not considered in this paper. For example, it may be possible that last level segments belonging to the same superior segment are stored near each other and cause a reduction in disk arm movement during their retrievals. This kind of refinements, which may well reverse our conclusions, require much more detailed analysis than that given in this paper.

V. CONCLUSIONS

The technique of evaluating alternate designs of file organizations based on the given file and application characteristics has been demonstrated through the example considered in this paper. Conventional hand calculations and imprecise rules of thumb become inadequate in dealing with complex problems of this magnitude. The usefulness of a quantitative evaluation tool such as the FOREM model can never be overemphasized.

The design alternatives, of course, represent only a small fraction of a wide variety of issues and tradeoffs encountered in designing any large file system. However, two observations should be noted here. First, in a complex design problem involving a large number of variables, it is not unusual that at each level of decision the resulting performance depends critically on the disposition of a small subset of variables but is relatively insensitive to the choice of the other variables over a wide range of values. Identification of the sensitivities of variables and their corresponding ranges of values would certainly be of positive value to the design process. Secondly, since the number of variables is large, an exhaustive enumeration of all possible combinations is prohibitively expensive even if each variable is assigned only a few trial values. It is thus imperative to devise some heuristic algorithms to reduce the number of combinations to be tested. The quantitative evaluation methodology presented in this paper represents an initial step in those directions.

ACKNOWLEDGMENT

The authors would like to thank M. Senko for many helpful discussions on the stratification of system abstractions and his suggestion of the term "the access path design level." The work is partially supported by the Rome Air Development Center through contract AF 30602-70-C-0114.

REFERENCES

(1) BUCHOLZ, W. File organization and addressing. IBM Systems Journal, 2 (June 1963) p. 86-111.

(2) GHOSH, S.P.; SENKO, M.E. File organization: on the selection of random access index points for sequential files. Journal of the ACM, 16:4 (October 1969) p. 569-579.

(3) TEICHROEW, D. A review of system design techniques. ISDOS Working Paper 17, University of Michigan, January 1969.

(4) McCUSKEY, W.A. Toward the automatic design of data organization for large scale information processing systems. Ph.D. thesis, Case Western Reserve University, January 1969.

(5) NUNAMAKER, JR., J.F. On the design and optimization of information processing systems. Ph.D. thesis, Case Western Reserve University, June 1961.

(6) LANGEFORS, B. Some approaches to the theory of information systems. BIT, 3 (1963) p. 229-254.

(7) DAY, R.H. On optimal extracting from a multiple file data storage system: An application of integer programming. Journal Op. Res., 13:3 (May-June 1965) p. 482-94.

(8) SENKO, M.E.; LING, H.; LUM, V.Y.; MEADOW, H.R.; BRYMAN, M.R.; DRAKE, R.J.; MEYER, B.C. File design handbook. Contract Report AF 30602-69-C-0100, November 1969.

(9) SENKO, M.E.; ABRAHAM, C.T.; GHOSH, S.P.; LUM, V.Y.; OWENS, P.W.; POMPER, I.H.; BAKER, F.T.; SCHENKEN, J.D.; WALKER, T.P. Formatted file organization techniques. Final Report AF 30602-4088, May 1967.

(10) SENKO, M.E.; LUM, V.Y.; OWENS, P.J. A file organization evaluation model - FOREM. 1968 IFIP Congress Proceedings, p. C19-23.

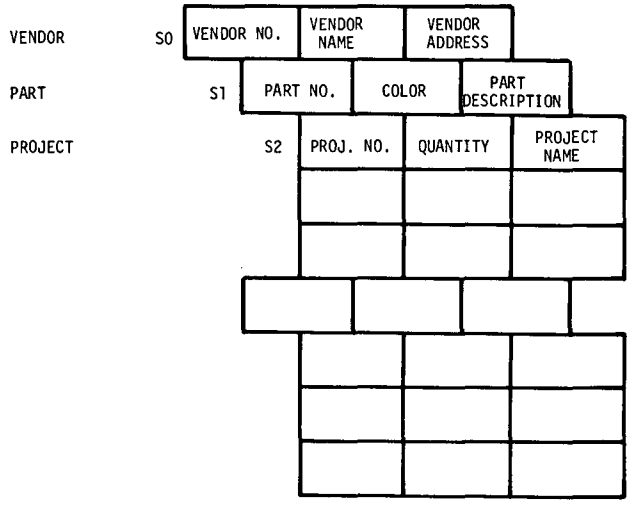
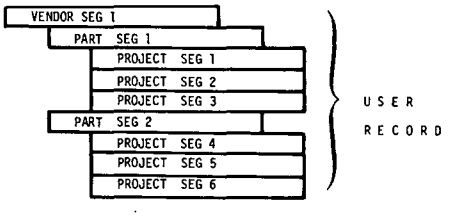


Figure 1. Schematic Representation of a User Record



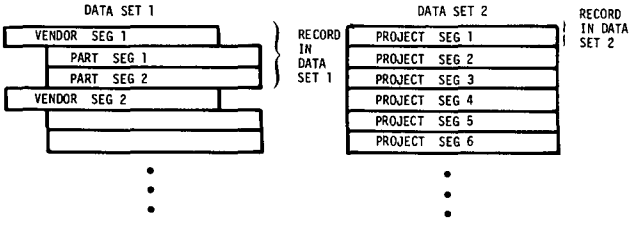


Figure 2. Assignment of Segments of a User Record to Data Set Records

Query No.	Description	Frequency
Q1	Given VENDOR NUMBER=1234, Retrieve Record*	9
Q2	VENDOR NAME=SMITH, PART NO.=01234, Retrieve Record	1
Q3	VENDOR NAME=SMITH, PART NO.=00012, LIST COLOR	1
Q4	VENDOR NAME=JONES, COLOR=RED, Retrieve Record	1
Q5	VENDOR NAME=APPLE, COLOR=BLUE, LIST VENDOR NO. & PART NO.	1
Q6	VENDOR NAME=APPLE, PROJECT NO.=2135, Retrieve Record	1
Q7	VENDOR NAME=SMITH, PROJECT NO.=4275, LIST VENDOR NO. and PARTS SUPPLIED	1
Q8	VENDOR NAME=JONES, Retrieve Record	1
Q9	COLOR=RED, PROJECT NO.=2135, LIST VENDOR NO. & PART NO.	1
Q10	PART NO.=00012, COLOR=BLUE, Retrieve Record	1
Q11	PART NO.=00012, Retrieve Record	1
Q12	PROJECT NO.=3124, LIST VENDOR NAMES SUPPLYING TO THAT PROJECT	1
Q13	Given VENDOR NO.=6337, Update COLOR	2
Q14	Given VENDOR NO.=2184, Update VENDOR ADDRESS & COLOR	2
Q15	Given VENDOR NO.=3526, LIST VENDOR NAME & COLOR	1
Q16	Given VENDOR NO.=7283, LIST PARTS SUPPLIED	1
Q17	Given VENDOR NO.=1124, Update Record	2

* RECORD here refers to the record format in Figure 1.

Table I

Query Number	Frequency	Processing Time In Seconds		
		Vendor Name Part No. Indexed Color Project No.	Vendor Name Indexed Color Project No.	Vendor Name Part No. Indexed Project No.
Q1	9	2.43	2.43	2.43
Q2	1	3.5	110	3.5
Q3	1	1.54	110	1.54
Q4	1	67.8	67.8	22.8
Q5	1	45.8	45.8	8.87
Q6	1	1.10	1.10	1.10
Q7	1	6.03	6.03	6.03
Q8	1	22.8	22.8	22.8
Q9	1	93.6	93.6	60.0
Q10	1	52.7	6896	21.4
Q11	1	21.4	686	21.4
Q12	1	39.6	39.6	39.6
Q13	2	3.14	1.86	1.24
Q14	2	3.92	2.60	2.60
Q15	1	0.27	0.27	0.27
Q16	1	0.27	0.27	0.27
Q17	2	7.81	6.50	6.50
Total Time:		373.17	8092.09	221.76

Table III

FIELD VALUE ITEM	FREQUENCY OF OCCURRENCE							
	CASE I				CASE II			
	Vendor Name	Part No.	Color	Proj. No.	Vendor Name	Part No.	Color	Proj. No.
V1	54	70	43,210	115	54	70	10,000	115
V2	150	210	31,566	70	150	210	10,000	70
V3	125	520	12,370	45	125	520	10,000	45
V4	80	305	12,854	375	80	305	70,000	375
V5	99,591	78	--	65	99,591	78	--	65
V6	--	1,003	--	913	--	1,003	--	913
V7	-	724	--	4,890	--	724	--	4,890
V8	--	97,090	--	9,672	--	97,090	--	9,672
V9	--	--	--	10,027	--	--	--	10,027
V10	--	--	--	12,300	--	--	--	12,300
V11	--	--	--	25,798	--	--	--	25,798
V12	--	--	--	35,642	--	--	--	35,642

In Table II the total occurrences in the Part No., Color, and Project No. fields have been normalized to 10^5. A multiplication factor of 2 should be used for Part No. and Color and a multiplication factor of 6 should be used for the Proj. No.

Table II

Query Number	Frequency	Processing Time In Seconds		
		Vendor Name Part No. Indexed Color Project No.	Vendor Name Indexed Color Project No.	Vendor Name Part No. Indexed Project No.
Q1	9	2.43	2.43	2.43
Q2	1	3.50	110	3.50
Q3	1	1.54	110	1.54
Q4	1	67.8	19.8	22.8
Q5	1	45.8	1L.9	8.82
Q6	1	1.10	1.10	1.10
Q7	1	6.03	6.03	6.03
Q8	1	22.8	22.8	22.8
Q9	1	93.6	27.6	60.0
Q10	1	52.7	2469	21.4
Q11	1	21.4	686	21.4
Q12	1	39.6	39.6	39.6
Q13	2	3.14	1.86	1.24
Q14	2	3.92	2.60	2.60
Q15	1	0.27	0.27	0.27
Q16	1	0.27	0.27	0.27
Q17	2	7.81	6.50	6.50
Total Time:		197.07	3522.2	221.76

Table IV

Query Number	Frequency	Processing Time In Seconds		
		One Data Set	Two Data Sets Both ISAM	Two Data Sets One ISAM, One DAM
Q1	9	2.43	11.2	4.17
Q2	1	3.50	12.4	3.24
Q3	1	1.54	1.49	1.49
Q4	1	19.8	179	99.5
Q5	1	16.9	44.6	44.6
Q6	1	1.10	1.10	1.10
Q7	1	6.03	3.52	3.52
Q8	1	22.8	181	63.66
Q9	1	27.6	74	58.2
Q10	1	19.5	93.2	62.1
Q11	1	21.4	92.4	37.5
Q12	1	39.6	34.5	14.7
Q13	2	3.14	3.14	3.14
Q14	2	3.92	3.92	3.92
Q15	1	0.27	0.18	0.18
Q16	1	0.27	0.18	0.18
Q17	2	7.81	10.6	8.51
Total Time:		373.17	730.63	409.71

Table V

ELEMENTS OF THE RANDOMIZED COMBINATORIAL FILE STRUCTURE

Richard A. Gustafson*
University of South Carolina, Columbia, South Carolina

ABSTRACT

A file structure designed to provide rapid, random access with minimum storage overhead is presented. Storage and retrieval are achieved by direct attribute combination-to-address transformation thereby negating the necessity for large file dictionaries or list-pointer structures. The attribute combination-to-address transformation is conceptually similar to key-to-address transformation techniques, but the transformation is not limited to operations on a single key but operates on the combination of several independent keys (or any subset of the combination) describing an item or request.

A storage and retrieval system utilizing the combinatorial file structure is developed. Storage and retrieval results derived from a simulated document library of 4000 items are presented. The new file organization is shown to have marked value with respect to minimum storage overhead and high retrieval speed.

KEY WORDS AND PHRASES

file structures, file organization, combinatorial file, key-to-address transformation, multiple attribute retrieval, file access method, data structures, hash-coding, attribute combination-to-address transformation, storage and retrieval system

I. Introduction

The nucleus of any information handling system is the file structure. All data in an information handling system flow from information sources into a structured file (collection of items of information) and ultimately to the information user. Effective management of this information flow has led to the application of mechanized, and in particular computer based, systems to alleviate the problems of information storage and retrieval. This application of automated information handling systems has placed a premium on the utilization of effective and efficient file structures. File effectiveness and efficiency may be measured with respect to the amount of storage required to store an information file and the speed with which information may be retrieved from the file. As one might suspect, storage efficiency and retrieval speed are normally at odds with one another.

Multiple-key file structures, presently in use with mechanized systems, range from simple unordered, linear files to sophisticated list and inverted files (4, 8, 9, 18, 23, 27). These files can be said to be typical and illustrate the diversity of storage requirements and retrieval speeds of the various contemporary file structures. On one hand is the linear file requiring storage only for the items and descriptor keys but necessitating a linear search of the entire file for any retrieval; while, on the other hand, are the inverted and threaded list files achieving rapid retrieval but requiring the storage overhead of indexes to the files in the form of dictionaries or pointers. This storage overhead can, in fact, become as large or larger, storage-wise, than the item file itself (8, 14).

The retrieval speed of the inverted and list files can, under certain conditions, also be disappointing (12, 14, 25). This is a result of the "term at a time" search of the inverted and list files which does not fully utilize the combinatorial properties of a multiple-key file. Ghosh and Lum (12, 25) have proposed file organizations designed to exploit key-combination retrieval but they require additional key-combination indexes and duplicate item pointer entries to accommodate all key combinations.

It would be ideal if the index, as a search mechanism, could be replaced by an algorithm (i.e., a short series of computation steps) which would operate on a request and produce the accession numbers of the items in the file relevant to the request. Such a search mechanism would be required to accept multiple-key requests (i.e., multiple index terms) and produce retrievals based upon any combination of terms. The remaining sections of this paper develop and present a search mechanism and file structure possessing these properties.

II. A Combinatorial Structure

The combinatorial properties of the storage and retrieval mechanisms can be easily illustrated graphically. With the assumption that each item can be described by three unique attributes taken from a descriptor set of five attributes, the num-

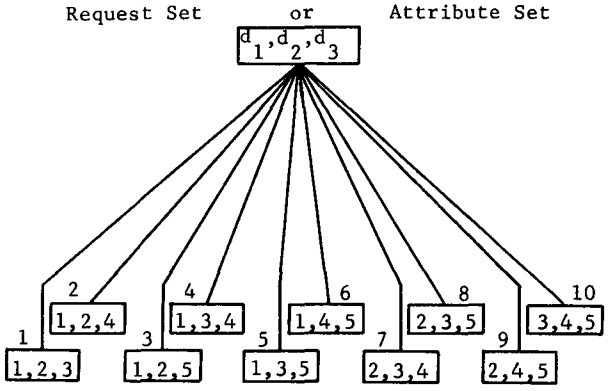
*Present Address: Air Force Technical Applications Center, Alexandria, Virginia

ber of possible attribute combinations can be calculated from the binomial coefficient:

$$\binom{N}{M} = \binom{5}{3} = \frac{5!}{3!(5-3)!} = 10$$

It can therefore be concluded that, in this example, any set of attributes describing an item must be one of ten possible combinations.

Representing the five possible attributes in the descriptor set by the numbers 1 through 5 yields the ten possible combinations illustrated in the following diagram.



Total Possible Combinations

From inspection of the above diagram it is seen that any possible set of three unique descriptors must map into one and only one of the possible combination "buckets".

$$g: \quad \{d_1,d_2,d_3\} \longrightarrow a: \qquad a \in \{1,2,3,4\ldots10\}$$

The term "bucket" is used as it is possible that two or more items could have the same attribute set; hence the word "bucket" is used to indicate the possibility of more than one item in a unique combination. The mapping, g, must produce an address, a, which is an element of the set $\{1,2,\ldots10\}$.

The Possible Combinations diagram allows the consideration of both storage and retrieval operations. The storage mechanism may be illustrated by means of the following example.

Item Storage

Item i	Attribute Set d_1,d_2,d_3	Mapped to Bucket a
1	2,3,5	8
2	2,3,4	7
3	1,2,3	1
4	1,3,4	4
5	1,3,5	5
6	3,4,5	10

The retrieval mechanism operates in a similar manner; however, all combinations containing the request attribute set must be addressed. The following Item(s) Retrieval example illustrates an obvious but important property of a combinatorial file structure. A particular request need only address some fraction of the combination buckets. The number of addressed buckets is a function of the number of specified attributes, L, where L must be greater than or equal to 1, and less than or equal to the number of attributes used to describe an item, M. M in this case is three.

Item(s) Retrieval

Number of Attributes in Request	Request Set	Possible Combinations	Mapped to Bucket
L	$d_1\ldots d_L$	d_1,d_2,d_3	a
3	1,3,4	1,3,4	4
2	1,3	1,2,3	1
		1,3,4	4
		1,3,5	5
1	3	1,2,3	1
		1,3,4	4
		1,3,5	5
		2,3,4	7
		2,3,5	8
		3,4,5	10

The number of addressed buckets may be determined from the binomial coefficient:

$$(1) \quad Q_{N,M,L} = \binom{N-L}{M-L} = \frac{(N-L)!}{(M-L)!\,(N-M)!} = \frac{(N-L)_{M-L}}{(M-L)!}$$

Where:

M - number of attributes/item
N - number of possible attributes
L - number of specified request attributes
($1 \leq L \leq M$)

$$(N-L)_{M-L} = (N-L)\ .\ (N-L-1)\ \ldots\ (N-M+1)$$

$Q_{N,M,L}$ - number of possible combinations for L specified attributes with fixed N and M

The fraction of combination buckets addressed by a request set consisting of L attributes will be termed the selection ratio. The selection ratio is defined by:

$$(2) \quad S.R._{N,M,L} = \frac{\text{Number of possible combinations for L specified attributes.}}{\text{Number of possible combinations for M attributes.}}$$

$$\frac{\binom{N-L}{M-L}}{\binom{N}{M}} = \frac{(M)_L}{(N)_L}$$

Where:

$$(M)_L = M \cdot (M-1) \ldots (M-L+1)$$
$$(N)_L = N \cdot (N-1) \ldots (N-L+1)$$

Equation (2) yields the simple but interesting result that the worse case selection ratio, L = 1, is:

$$S.R._{N,M,1} = \frac{M}{N}$$

Application of equations (1) and (2) to the case of five possible attributes with three attributes per item with requests of one, two, and three attributes yields respectively:

$$N = 5;\ M = 3;\ \binom{N}{M} = 10$$

L	$Q_{5,3,L}$	$SR_{5,3,L}$
1	6	3/5
2	3	3/10
3	1	1/10

which is in complete agreement with the figures deduced from the possible combinations diagram for the same case.

Combinatorial Mapping

It would appear that the combinatorial structure offers the ideal mechanism of storage and retrieval. That is to say, a request defines a volume of the address space within which all stored items are pertinent to the specified request and outside of which the items are irrelevant to the request. The combinatorial structure does, however, present an addressing or mapping problem.

The required mapping is identical to the desired mapping for the combinatorial space as presented in conjunction with the Possible Combinations diagram.

$$(3)\quad g: \left\{d_1, d_2 \ldots d_M\right\} \longrightarrow a: \quad a \in \left\{1, 2 \ldots P\right\}$$

Where:

$$P = \binom{N}{M}$$ (Number of possible combinations in the combinatorial address space)

The combinatorial mapping g, must collapse M indices or attributes into a single index, the bucket address.

The mapping g, may be developed in a fairly straightforward manner by first considering how all possible combinations of N objects taken M at a time without replications and with unordered samples may be formed. There are, of course, $(N)_M/M!$ possible combinations. These combinations, each consisting of M indices (attributes), may be generated by varying the indices in the following manner:

$$(4)\quad \left\{d_{i_1}, d_{i_2} \ldots d_{i_M}\right\}$$

$i_1 = M, M+L \ldots N$

$i_2 = M-1, M, \ldots (i_1-1)$

.

.

.

$i_M = 1, 2 \ldots (i_{M-1}-1)$

Where d_j is the j^{th} element of the object set or, in this case, the j^{th} attribute or index in the descriptor set. If the M^{th} index in the attribute set, $\left\{d_{i_1}, d_{i_2} \ldots d_{i_M}\right\}$, is varied most rapidly, and the $(M-1)^{th}$ varied next, through the first index varying least rapidly, all possible combinations will be produced in a fixed order (thereby effectively deleting permutations in any combination) and no replications in any combination will occur.

Since all possible attribute sets can be generated in a fixed and mathematically predictable manner, it would appear that the location or position of a particular attribute set in the array of possible attribute sets could be determined. Indeed, this is the case (14, 24).

To determine the location of a specified attribute set in the array of possible attribute sets as produced by the possible-combinations generator, one may answer either of two questions. How many combinations are below the specified set; or alternately, how many are above? The former question will be answered as the development is straightforward.

Given an attribute set consisting of M elements ranked from highest to lowest by numeric value, $\left\{d_1', d_2' \ldots d_M'\right\}$, it is certainly true that all attribute combinations formed by taking M attributes at a time from a set of $(d_1'-1)$ attributes (note that d_1' is the largest value in the attribute set) must be below the given set. Likewise, it may be stated that all combinations formed by fixing d_1' in each combination and taking (M-1) attributes from a set of $(d_2'-1)$ attributes to complete the combinations must also be below the given attribute set in the array of possible-combination sets. This argument may be extended to the d_M' element where $d_1', d_2' \ldots d_{M-1}'$ are all fixed in each combination and the combinations are completed by taking M-(M-1) or 1 attribute from a possible set of $(d_M'-1)$ attributes.

The number of ways M attributes may be taken from a set of $(d_1'-1)$ attributes without replication and ignoring order is simply:

$$\binom{d_1'-1}{M}$$, a binomial coefficient.

It may therefore be concluded that the number of possible combinations below a specified attribute set is:

$$\binom{d_1'-1}{M} + \binom{d_2'-1}{M-1} + \ldots \binom{d_M'-1}{1}$$

where: $d'_1 > d'_2 > d'_3 \ldots > d'_M$. Adding one to the number of combinations below a specified attribute set yields the location of that attribute set in the array of possible combinations as produced by the possible-combinations generator. This location is the "bucket address" in the address space. The mapping, g, is therefore:

$$(5)\quad g: \left\{d'_1 d'_2 \ldots d'_M\right\} \longrightarrow a: \quad a \in \left\{1,2,3 \ldots P\right\}$$

$$\text{where: } P = \binom{N}{M}$$

$$g(d'_1, d'_2 \ldots d'_M) = a = \binom{d'_1 - 1}{M} + \binom{d'_2 - 1}{M-1}$$

$$+ \ldots \binom{d'_M - 1}{1} + 1$$

where the primes on the d's simply indicate ordering from highest to lowest.

Although the preceding development of the mapping, g, might appear overly restrictive, it is in fact quite general. With the stipulation that the descriptor set of possible attributes is reduced to the subscripts of the actual attributes ranging from 1 to N, the mapping as defined by (5) is perfectly correct.

Retrieval Operation

While the preceding development of the attribute set to address mapping defines the storage mechanism, a retrieval operation does not necessarily involve a single mapping between the request attribute set and the address space. This property of the retrieval mechanism was discussed earlier and was illustrated by equation (1) which states that Q mappings are required where:

$$Q = \frac{(N-L)_{M-L}}{(M-L)!}$$

It is to be recalled that N and M are, respectively, the number of possible attributes and the number of attributes per item. L is the number of attributes in a request ($1 \leq L \leq M$).

As a request must search the Q buckets contained in the fraction of the volume of the address space as defined by the request, one method of mapping to these buckets would be to generate all possible combinations of attribute sets containing the request attributes and map to the address space one to one for each possible combination. As this technique offers conceptual simplicity, it will be pursued.

An obvious starting point in the development of a retrieval possible-combinations generator is the total possible-combinations generator as previously presented in (4). With the consideration of (4), the retrieval-combinations problem reduces to the generation of all possible combinations of (N-L) things taken (M-L) at a time. This may be accomplished by defining a modified descriptor set for each request consisting of the original descriptor set with the L request attributes deleted. The Q combinations then formed by the combinations generator yield all possible combinations of M attributes relevant to the request when the L original request attributes are included in each combination. These attribute combinations may then be mapped to the Q buckets by the mapping, g. This mapping operation is:

$$(6)\quad g: \left\{d'_{i_1}, d'_{i_2} \ldots d'_{i_M}\right\} \longrightarrow a_i: \quad a_i \in \left\{1,2 \ldots P\right\}$$

$$\text{Where: } P = \binom{N}{M}; \quad i = 1,2 \ldots Q$$

$$\text{and: } Q = \binom{N-L}{M-L}$$

Examination and retrieval of the bucket items complete the retrieval operation.

In that the combinatorial file structure stores and retrieves each item as a function of its several attributes without resorting to complex list structures, the combinatorial structure closely approximates associative memory processing by "software" address decoding (7, 21). This characteristic is basic to all information storage and retrieval systems but in this case, is exhibited on the file structuring level.

III. A Randomized Combinatorial Structure

Although the combinatorial file structure may be pleasing from the standpoint of a storage and retrieval mathematical model, it has limited application in its present form. In order to preserve associations among items and to assign each item in the file a unique address, all possible combinations of descriptors, subject to a priori constraints, could be required to map into unique addresses. Should this requirement be levied, the preceding development of the combinatorial file structure is of little use; as, for any moderately large number of possible descriptors with more than a few attributes per item, the number of possible combinations becomes far too large for any practical application. This conclusion is substantiated by consideration of the number of combinations, 7.5×10^7, of only one-hundred descriptors taken five at a time.

The "possible combinations" problem is not unique to the combinatorial structure, but is encountered in most information storage systems. The classical example is that of using names for record (the basic information unit, e.g., payroll entry, account entry, etc.) identification. As the number of possible names is far greater than the normal record file, techniques for mapping the names to unique locations in the storage media have been developed (16, 22, 28, 31, 34, 35). One of the more popular and efficient methods of accomplishing this mapping is to randomize the record "names" and normalize the random codes to the file length. Each record may then be stored in its randomized location. The "name" randomization may be accomplished by performing arithmetic operations on the internal machine code used to represent the name, i.e., hash-coding.

Although the techniques of randomized file

addressing are varied and the associated problems interesting, of principle interest is the application of the randomizing concept to the combinatorial file structure. Whereas randomized file addressing is concerned with the one to one mapping of a record identification name to a storage address, the combinatorial structure requires not only that a combination of attributes, which forms a record identification, map to a storage address but, additionally, that any subset of the attributes forming the identification maps one to Q where Q defines the number of mapped addresses. It is therefore required that any technique of randomizing aimed at reducing the number of possible combinations retain the combinatorial mapping properties of the combinatorial structure (i.e., attribute subset mappings of one to Q must not be sacrificed by randomization).

Examination of the binomial coefficient defining the number of possible combinations which may be formed from N attributes taken M at a time yields an insight into possible techniques to alleviate the "possible combinations" problem. It is readily apparent that the reduction of either N, the number of possible attributes, or M, the number of attributes per sample, will reduce the number of possible combinations (this is true with the restriction that $N > 2M$). The randomization of the M attributes forming an item description, however, would annul the independence of the M attributes and would therefore invalidate a combinatorial mapping scheme when retrieval based upon a partial description (L out of M attributes) is indicated. Although the randomization of attributes on an item description level must be ruled out as combinatorial properties are destroyed, the randomization of the descriptor universe consisting of all N possible attributes is possible while still retaining the combinatorial structure.

Randomization Technique

The randomization technique consists of mapping a subset of the elements of a set D onto the elements of a set C such that the number of elements in set C is less than and normally much less than the number of elements in set D. Or in set notation

$$f: \{d_1, d_2 \ldots d_M\} \rightarrow \{c_1, c_2 \ldots c_M\}$$

where $\{d_1, d_2 \ldots d_M\}$ is a subset of length M of set D being mapped into M elements, $\{c_1, c_2 \ldots c_M\}$, of set C. Normally, the descriptor set, D, of all possible attributes consists of word descriptions in English while the randomized descriptor set, C, may be of any arbitrary representation. It may therefore be concluded that with an a priori knowledge of the representation of set D, a mapping, f, may be devised which, as a result of its operation on an element of set D, produces an element of set C in a convenient representation. The statement "in a convenient representation" is stressed as the combinatorial mapping as developed in the previous chapter operates on the subscripts of the attributes. Naturally, it would be very "convenient" to require that the randomized descriptor set, C, consist of subscripts (integers ranging from 1 to N).

The mapping necessary to achieve randomization of M attributes describing an item is, therefore, in set notation:

$$(7) \quad f: \{d_1 d_2 \ldots d_M\} \longrightarrow \{c_1, c_2 \ldots c_M\}: \quad c_i \in \{1, 2 \ldots N\}$$

The randomized code set, C, may be of any length, N, necessary to reduce the possible combinations to manageable levels and the mapping is determined by the digital storage device characteristics.

Duplicate Code Probability

Some immediate consequences of the randomization mapping are apparent. In that any possible descriptor must map to one of the integer elements of the randomized code set and all storage and retrieval operations use these mapped integers, the randomized combinatorial structure is functionally independent of the file descriptor universe. Another, perhaps detrimental, aspect of the randomization of the descriptors is the possibility of two or more descriptors mapping to the same randomized code. This possibility of duplicate codes requires a modification to the combinatorial structure.

The previous section developed the combinatorial mapping structure based upon the assumption of M unique attributes describing an item. This assumption appeared valid in that the duplication of an attribute does not add additional information to an item description. The randomization mapping, however, negates the unique attribute assumption.

In order to better evaluate the impact of possible duplicate randomized codes on the combinatorial structure, it is necessary to form a more definitive measure of this duplication than simply the word "possible". With the assumption that the randomization function produces a truly random mapping of the attribute set of length M, resulting in a randomized code set of length M, a measure of the "degree" of code duplication reduces to finding the probability of obtaining J unique codes in a sample of M codes formed by drawing M codes from a universe of N distinguishable codes with replacement. This probability determination is similar to the classical "birthday problem"; however, it is necessary not only to determine the probability that no two codes in the sample are the same (M unique codes), but also to determine the probability of M-1, M-2...1 unique codes in a sample of M codes. This information is necessary in that each duplication of a code effectively reduces the number of attributes describing an item by one. The degree of this information loss is determined by the following (10, 14).

The probability that in a sample formed by drawing M items from N items with replacement, any of the N items may appear K_1 times while the remaining N-1 items appear $K_2, K_3 \ldots K_N$ times is:

$$(8) \qquad P = \frac{\frac{M!}{K_1!K_2!\ldots K_N!} \cdot \frac{N!}{1_1!1_2!\ldots 1_M!}}{N^M}$$

where:

M - number of items in sample

N - total number of items

K_i - total number of times that the i^{th} item appears in the sample

$(K_1 + K_2 \ldots K_N = M)$

1_i - total number of times that the i^{th} digit appears in the set of K's.

$(0 \leq i^{th}$ digit $\leq M$: i^{th} digit is integer)

$(1_1 + 1_2 \ldots 1_M = N)$.

The probability of finding M unique items in a sample of M items is therefore:

$$P = \frac{\frac{M!}{1!1!..1!0!..0!} \cdot \frac{N!}{M!(N-M)!0!}}{N^M}$$

$$= \frac{N\cdot(N-1)\ldots(N-M+1)}{N^M} = \frac{(N)_M}{N^M}$$

as: $K_1, K_2 \ldots K_M = 1$ and $K_{M+1}, K_{M+2} \ldots K_N = 0$, and

$1_1 = M$, $1_2 = N-M$ and $1_3 1_4 \ldots 1_M = 0$,

which reduces to the expected solution of the "birthday problem".

In that the probability of J unique items appearing in a sample of size M is not always determined by a single combination of duplications (the K set), all possible combinations of duplications resulting in a specified number of unique items must be listed and their probabilities evaluated and summed to yield the probability of J unique items in a random sample. The next section will contain an application of formula (8) which will further illustrate all "possible duplication combinations".

Duplicate Code Handling

Because of the possibility of duplicate attribute codes arising due to the randomization mapping of (7), it is necessary to devise some method capable of handling duplicate attribute codes in the context of combinatorial mapping. One possible solution to this duplication problem would be to redevelop the combinatorial structure allowing the duplication of attributes. In view of the preceding development, this would not be particularly difficult; however, a simple fact illustrates the futility of a combinatorial structure with the possibility of duplicate attributes: all combinations in a duplicate attribute combinatorial structure are not equally likely.

Another, more attractive, solution to the duplicate attributes problem would be to ignore duplicate attributes and, for the purposes of the storage mapping, generate at random non-duplicate attribute codes to replace any duplicates. Although this obviously constitutes a loss of retrieval efficiency, degradation occurs only at the search level as, although a set of requestors could possibly encompass a large area of the file, the requestors would, of course, define the area into which the original attribute set had mapped. It is to be again noted that the duplication of attributes is a probabilistic function as defined by (8) and the action of randomizing duplicates would be taken only on occasion. Elaboration on the probabilistic nature of randomized attribute duplication will be given in the next section.

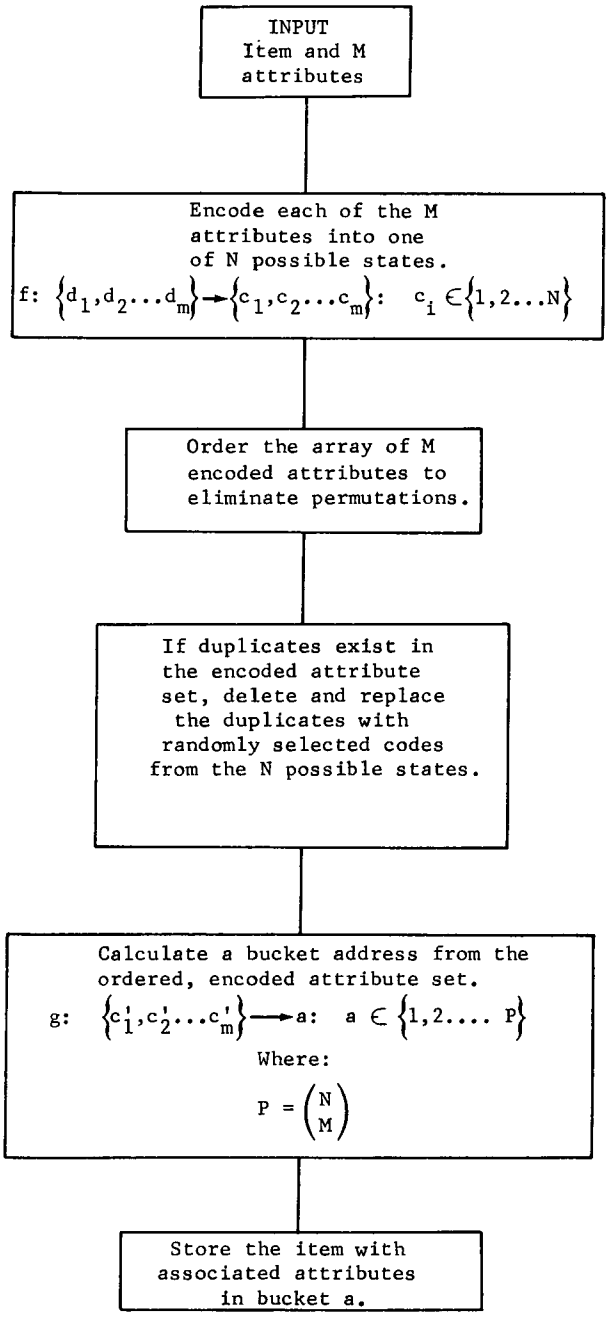
The random replacement of duplicate attribute codes as well as the normal randomization of the original attributes necessitates a search for original descriptor/requestor attribute matches subsequent to bucket address decoding during retrieval operation. This search necessity is a result of the attribute randomization phase (encoding) where mapping of original attributes is many to one. This simply means that a mapped bucket address may or may not contain the original requestor attributes, hence, the need for the search operation. It is hastened to indicate that all unmapped buckets will not contain the request attributes. The retrieval operation may therefore be perhaps best described as a "divide and conquer" technique.

IV. File Design and Application

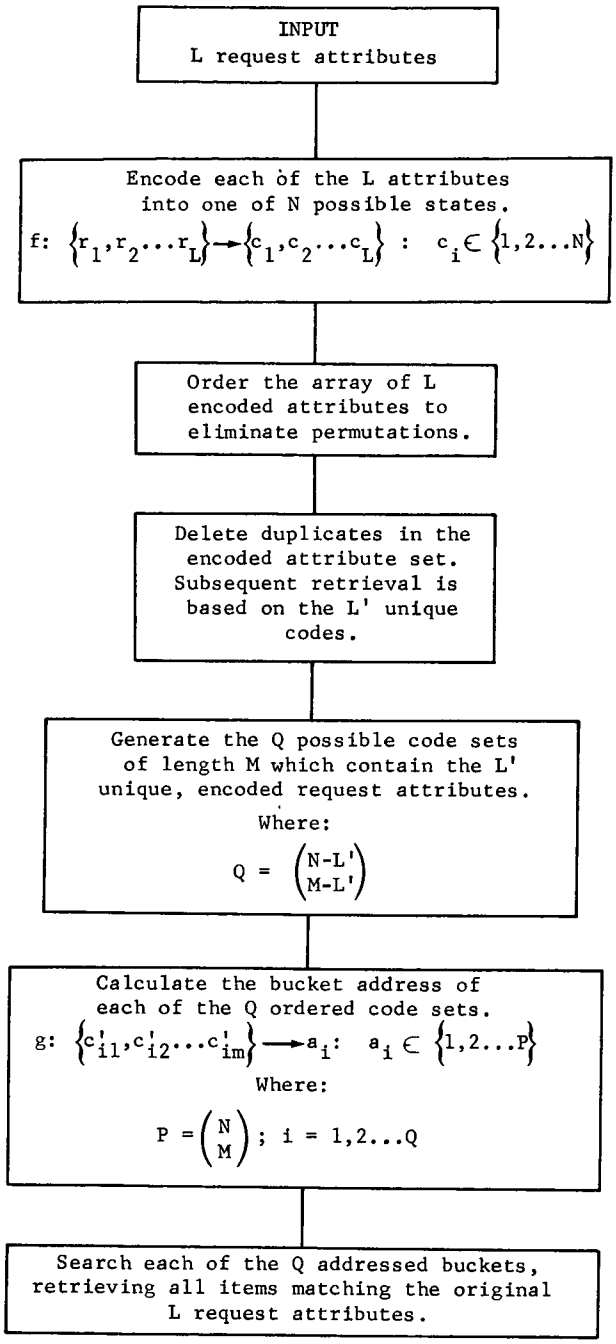
In order to demonstrate the feasibility of the Randomized Combinatorial File Structure and to illustrate the techniques and considerations of a system design utilizing this file structure, an information storage and retrieval simulator was designed. This simulator encompassed all of the randomized combinatorial file structuring concepts and the system design closely followed the item storage and retrieval block diagrams as presented on the following page. It is to be noted that the choice of the term "simulator" over the term "system" is made simply to indicate that no attempt was made to optimize instruction coding, an artificial data base was used, and core storage was used as the information storage media.

The first step in the design of a simulator is to select the combinatorial file structure parameters. The basic parameters are those of the number of descriptor attributes per item, M, and the total number of possible attribute codes, N. The former is dictated by the item description requirements while the latter is selected by maximum retrieval efficiency versus required file size considerations.

In order to illustrate the dependence of the selection ratio (retrieval efficiency) and the number of possible attribute code combinations (a consideration dependent on the required file size) on N, a computer program was devised to produce a


INPUT
Item and M attributes
Encode each of the M attributes into one of N possible states.
f: $\{d_1,d_2\ldots d_m\} \rightarrow \{c_1,c_2\ldots c_m\}$: $c_i \in \{1,2\ldots N\}$
Order the array of M encoded attributes to eliminate permutations.
If duplicates exist in the encoded attribute set, delete and replace the duplicates with randomly selected codes from the N possible states.
Calculate a bucket address from the ordered, encoded attribute set.
g: $\{c'_1,c'_2\ldots c'_m\} \rightarrow a$: $a \in \{1,2\ldots P\}$
Where:
$P = \binom{N}{M}$
Store the item with associated attributes in bucket a.

ITEM STORAGE BLOCK DIAGRAM


INPUT
L request attributes
Encode each of the L attributes into one of N possible states.
f: $\{r_1,r_2\ldots r_L\} \rightarrow \{c_1,c_2\ldots c_L\}$: $c_i \in \{1,2\ldots N\}$
Order the array of L encoded attributes to eliminate permutations.
Delete duplicates in the encoded attribute set. Subsequent retrieval is based on the L' unique codes.
Generate the Q possible code sets of length M which contain the L' unique, encoded request attributes.
Where:
$Q = \binom{N-L'}{M-L'}$
Calculate the bucket address of each of the Q ordered code sets.
g: $\{c'_{i1},c'_{i2}\ldots c'_{im}\} \rightarrow a_i$: $a_i \in \{1,2\ldots P\}$
Where:
$P = \binom{N}{M}$; i = 1,2...Q
Search each of the Q addressed buckets, retrieving all items matching the original L request attributes.

ITEM RETRIEVAL BLOCK DIAGRAM

table listing the number of possible attribute combinations (buckets) for N ranging from four to one hundred while M ranged from three to ten. In addition to the number of buckets, the table also listed the selection ratio and the number of addressed buckets for one out of M to M out of M retrieval attributes for each combination of N and M. Although this table is far too lengthy for inclusion in this paper, a portion of these data is presented in graphical form on Graphs 1 and 2. Both graphs cover only the narrow range of N from 6 to 20 and M from 4 to 6 which is of importance to the simulator design.

Graph 1 presents a plot of the number of buckets against the number of possible attribute codes, N, for various values of M, the number of attributes per item. Graph 2, on the other hand, presents the worst case selection ratio plotted against N for various values of M. Inspection of Graph 1 yields the fairly obvious observation of the rapid rise of the number of buckets due to an increase in N for a fixed M. This rapid increase in the number of buckets places a limit on the selection of N for a fixed M as discussed in the previous section. Graph 2 illustrates the decrease in the worst case selection ratio as N (possible codes) is increased for a fixed M. From the standpoint of retrieval efficiency, therefore, it is advantageous to select an N as close as possible to the limit as dictated by Graph 1.

The choice of core storage as an information storage medium precluded the use of a very large data base. In view of this constraint, an artificial data base of 4000 items with five descriptor attributes per item was formed. With the assumption of 4000 items with five descriptor attributes per item, the selection of N may be made.

Reference to Graph 1 with M = 5 indicates that N must be below 15 to yield fewer than 4000 buckets. In order to allow for the random distribution of items in the buckets, N = 14 is chosen. This choice yields 2002 buckets with a reasonable likelihood that any examined bucket will contain at least one item after a uniform random distribution of the items among the buckets.

The choice of N = 14 yields a worst case selection ratio of .357 as seen from inspection of Graph 2. The remaining selection ratios with the number of buckets addressed by each are presented in the table below.

Known Attributes (Requestors)	Selection Ratio	Addressed Buckets
1 out of 5	.35714	715 out of 2002
2 out of 5	.10989	202 out of 2002
3 out of 5	.02747	55 out of 2002
4 out of 5	.004995	10 out of 2002
5 out of 5	.004995	1 out of 2002

SELECTION RATIOS

(14 Possible Attribute Codes with 5 Attributes per Item)

Consideration of the preceding table illustrates the "divide and conquer" retrieval operation of the combinatorial file structure.

Effect of Duplicate Codes

Although the preceding table presents a true picture of the retrieval efficiency from the attribute code level, the effect of possible duplicate codes must be treated. A table presenting the probabilities of no duplicates of requestor attributes is shown below. The detail of possible duplications may be derived from Equation 8 and is not presented.

Request Attributes	Probability of No Duplications
5	.44669
4	.62536
3	.79592
2	.92857
1	1.00000

PROBABILITY OF NO DUPLICATE CODES
(Drawn from 14 Codes)

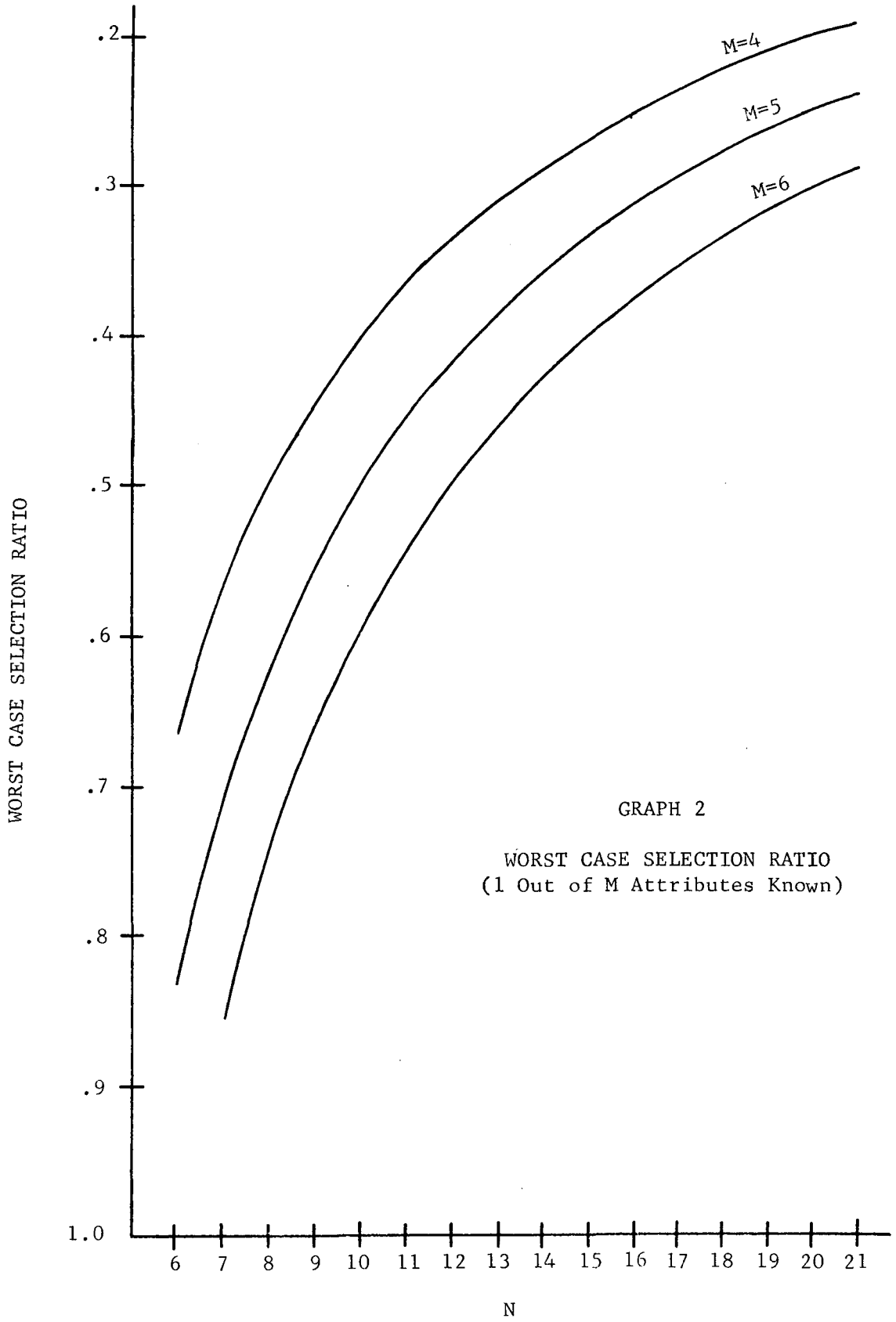
It comes as no surprise that the probability of duplicate codes increases as the number of specified request attributes increases. This increase, however, may be considered as a fairly small perturbation to retrieval efficiency in view of the far more rapid decrease in the selection ratio as the number of specified attributes increases (see Selection Ratios Table previously presented).

Implementation and Results

The combinatorial file structure simulator, comprised for the most part of Fortran IV routines, was implemented on an IBM 7040 computer and storage and retrieval tests made. These test runs were performed by first utilizing a storage mode programming package to read a prepared tape of 4000 descriptor described items which were then stored in a manner consistent with the combinatorial file structure; and secondly, by utilizing a retrieval mode package to read card input requests consisting of various numbers of requestors and to retrieve all items in the combinatorial file which were fully or partially described by each request.

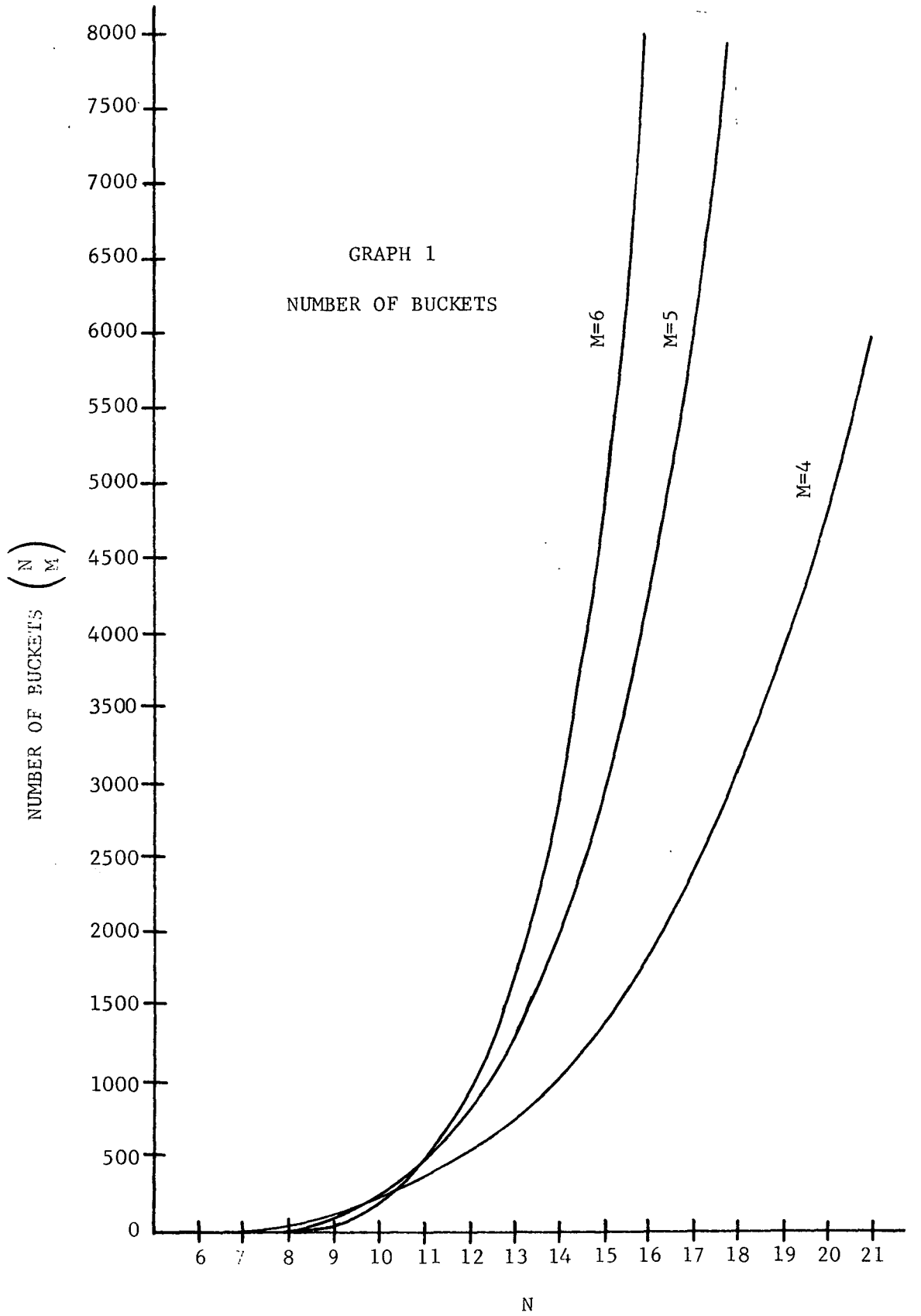
The retrieval times for a typical retrieval test are presented in the following table. This test consisted of five requests with the requests ranging from five to one requestor attributes respectively. Each request is listed with the number of request terms, time per bucket addressed, and the total time used to satisfy the request.

Although the time used to satisfy each request should not be considered as an evaluation of retrieval efficiency (e.g., no attempt was made to isolate input/output operations from the total time), the times are of interest when considered from a "relative time" standpoint. If the opera-



GRAPH 2

WORST CASE SELECTION RATIO
(1 Out of M Attributes Known)


GRAPH 1
NUMBER OF BUCKETS
NUMBER OF BUCKETS $\binom{N}{M}$
8000
7500
7000
6500
6000
5500
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
N
M=6
M=5
M=4

tion of retrieval mode simulator was as predicted, the "time to retrieve" should have been in direct proportion to the predicted number of buckets each request should address. This simply indicates that "time per bucket" should be relatively constant. This is indeed the case, and is illustrated in the table.

Request Terms	Time to Retrieve (Seconds)	Addressed Buckets (Predicted)	Time Per Bucket (Milliseconds)
5	.07	1	70
4	.28	10	28
3	1.42	55	26
2	5.50	202	27
1	17.85	715	25

As can be seen, all of the requests (with the exception of the first) exhibited an average "time per bucket" of 25 to 28 milliseconds. The discrepancy in the time of the first request may be attributed to input/output overhead as only one bucket is addressed and the time presented is near the resolution of the computer interval timer (16.67 milliseconds).

V. Conclusion

The randomized combinatorial file structure achieves storage and retrieval mapping by direct combinatorial address decoding. This direct mapping property negates the necessity for the storage of large file directories or list structures (e.g., an inverted file directory) while retaining the random access, rapid retrieval characteristics of list structured files (4, 8, 9, 18, 23, 27). The combinatorial mapping is conceptually similar to key-to-address transformation techniques (16, 22, 28, 31, 34, 35) but the transformation maps not as a function of a single key but as a function of a combination of keys.

Although the application of the file structure presented in this paper was made on a limited scale, implementation of the structure at a significantly greater level can be envisioned. In order to accommodate a large library of data, auxiliary random access storage such as disk files or auxiliary large capacity core storage could be utilized for the storage of items of information. The system design techniques presented in the application section are applicable to a large-scale system and should be easily integrated in an information storage and retrieval system.

BIBLIOGRAPHY

(1) ARON, J. Information systems in perspective. Computing Surveys, 1:4 (December, 1969) 213-236.

(2) BECKER, J.; HAYES, R. Information storage and retrieval: tools, elements, theories. Wiley, New York, 1963.

(3) BOURNE, C. Methods of information handling. Wiley, New York, 1963.

(4) CANTER, J.D.; DONAGHEY, C. E. UPLIFTS-University of Pittsburgh linear file tandem system. Communications of the ACM, 8:9 (September, 1965) 579-581.

(5) CHERRY, C.; ed. Information theory. Butterworth, London, 1961.

(6) CHEYDLEUR, B.; ed. Colloquium on technical preconditions for retrieval center operations. Spartan Books, Washington, D. C., 1965.

(7) CHEYDLEUR, B. F. SHIEF: a realizable form of associative memory. American Documentation, 14:1 (January, 1963) 56-57.

(8) DODD, G. Elements of data management systems. Computing Surveys, 1:2 (June, 1969) 117-133.

(9) DZUBAK, B. J.; WARBURTON, B. J. The organization of structured files. Communications of the ACM, 8:7 (July 1965) 446-452.

(10) FELLER, W. An introduction to probability theory and its applications. Wiley, New York, 1963.

(11) GARVIN, P.; ed. Natural language and the computer. McGraw-Hill, New York, 1963.

(12) GHOSH, S.; ABRAHAM C. Application of finite geometry in file organization for records with multiple-valued attributes. IBM Journal of Research and Development, 12:2 (March, 1968) 180-187.

(13) GOLOMB, S.; BAUMERT, L. Backtrack programming. Journal of the ACM, 12:4 (October, 1965) 516-524.

(14) GUSTAFSON, R. A randomized combinatorial file structure for storage and retrieval systems. Ph.D. thesis, University of South Carolina, Columbia (December, 1969).

(15) GUTENMAKHER, L. Electronic information-logic machines. Interscience, New York, 1963.

(16) HANAN, M.; DALERMO, F. P. An application of coding theory to a file address problem. IBM Journal of Research and Development, 7:2 (April, 1963) 127-129.

(17) HAYS, D. Introduction to computational linguistics. Elsevier, Amsterdam, 1967.

(18) HSIAO, D.; HARVARY F. A formal system for information retrieval from files. Communications of the ACM, 13:2 (February, 1970) 67-73.

(19) KENT, A. Textbook on mechanized information retrieval. Interscience, New York, 1962.

(20) KENT, A.; TAULBEE, O.; eds. Electronic information handling. Spartan Books, New York, 1965.

(21) KISEDA, J. R.; PETERSEN, H. E.; SEELBACH, W. C.; TEIG, M. A magnetic associative memory. IBM Journal of Research and Development, 5:2 (April, 1961) 106-121.

(22) KOHNEIM, A. G.; WEISS, B. An occupancy discipline and applications. SIAM Journal on Applied Mathematics, 14:6 (November, 1966) 1266-1274.

(23) LEFKOVITZ, D. File structures for on-line systems. Lecture notes for ACM Seminar on File Structures. Detroit, 1967.

(24) LEHMER, D. The machine tools of combinatories. In: E. Beckenbach; ed. Applied Combinatorial Mathematics. Wiley, New York, 1964.

(25) LUM, V. Y. Multi-attribute retrieval with combined indexes. Communications of the ACM, 13:11 (November, 1970) 660-665.

(26) MEADOW, C. Man-machine communication. Wiley, New York, 1970.
(27) MEADOW, C. The analysis of information systems. Wiley, New York, 1967.
(28) PETERSON, W. W. Addressing for random-access storage. IBM Journal of Research and Development, 1:2 (April, 1957) 130-146.
(29) SALTON, G. Automatic information organization and retrieval. McGraw-Hill, New York, 1968.
(30) SALTON, G. Data manipulation and programming problems in automatic information retrieval. Communications of the ACM, 9:3 (March, 1966).
(31) SCHAY, G.; RAVER, N. A method for key-to-address transformation. IBM Journal of Research and Development, 7:2 (April, 1963) 121-126.
(32) SCHECTER, G.; ed. Information retrieval: a critical view. Thompson, Washington, D. C., 1967.
(33) SPIEGEL, J.; WALKER, D.; eds. Information systems sciences: proceeding of the second congress. Spartan Books, Washington, D. C., 1965.
(34) TARTER, M.; KRONMAL R. Non-uniform key distribution and address calculation sorting. In: Proceedings of 21st National Conference, ACM. Thompson, Washington, D. C., 1966.
(35) TOYODA, J.; TEZUKA, Y.; KASHARA, Y. Analysis of the address assignment problem for clustered keys. Journal of the ACM, 13:4 (October, 1966) 526-532.
(36) VICKERY, B. On retrieval system theory. Butterworth, London, 1965.
(37) WALSTON, C. E. Information retrieval. In: F. Alt; M. Rubinoff; eds. Advances in Computers. Academic Press, New York, 1965.

A BALANCED TREE STORAGE AND RETRIEVAL ALGORITHM

Gary D. Knott
Division of Computer Research and Technology
National Institutes of Health
Bethesda, Maryland

ABSTRACT

A storage and retrieval scheme which places items to be stored at the nodes of a binary tree is discussed. The tree is always balanced in a certain sense thus insuring that no excessively long search paths can exist. In addition to presenting the storage and retrieval algorithms, the deletion problem is also solved. The programming approaches involved yield a non-trivial case study of list-processing techniques. Finally, a cost analysis is given.

KEY WORDS AND PHRASES

storage and retrieval, searching, data structures, balanced trees

1. Introduction

Below we describe a storage and retrieval method using a data structure which we shall call a balanced tree. This scheme was given by C. C. Foster in [4], where he attributes it to G. M. Adel'son-Vel'skij and Y. M. Landis [1]. It has been used by David E. Ferguson [3] as a means for the organization of symbol tables in assemblers. The deletion procedure given later, as well as the determination of an upper bound for the number of compares required to search n items organized in a balanced tree, are due to comments by Donald E. Knuth. George A. Miller contributed significantly to the algorithms given here by uncovering several bugs.

The method of storing randomly-received items as nodes in a binary tree such that x is a left son of y only if $x<y$, and x is a right son of y only if $y \leq x$ is well-known [2, 6, 8, 9]. Given such a tree and an item key, the search time (number of 3-way compares) required to find a matching item is generally a logarithmic function of n, where n is the number of items or nodes in the given tree.

However, a search time which is necessarily a logarithmic function of n occurs only when the given tree is balanced, that is, when no excessively short and long paths exist. Such paths tend to increase the number of compares required on the average.

The scheme described below controls the structure of the tree which is generated by dynamically adjusting it, if required, to insure it will, in fact, always be balanced. These adjustments increase the update costs, but they insure that search costs are always bounded by a logarithmic function of n. Furthermore, the increase in update cost is modest and may often be less than actual update costs in an unbalanced tree, due to the increased search time which may be required there.

2. The Underlying Data Structure

We shall take as our basic data structure an area D composed of four vectors (V, L, R, B). Thus D may be considered as a matrix of four columns, or alternatively, as a collection of four-component row vectors. A node is represented by a particular quadruple $(V_i, L_i, R_i, B_i) = D_i$. V_i is the value of the node; in general we think of V_i holding an item, or a pointer to an item contained in the set of items to be stored and retrieved. L_i is an index to the left son node, D_{L_i}, of the current node, or zero if no left son exists. R_i is similarly an index to the right son node, D_{R_i}, of the current node, or zero if no right son exists. B_i is an integer value such that $|B_i| \leq 1$. B_i is called the balance of the tree or sub-tree whose root is the current node. B_i is defined more precisely below.

f is defined as an index to D such that D_f is the first empty quadruple in D. We shall assume that initially f is properly set. Moreover, V_f is an index to the next free quadruple in D, and so on. Thus we have a standard free-space list organized in D with f taken as the head of the list. If the quadruple D_a is the last node of the free-space list, then $V_a = 0$.

The root node of the entire tree stored in D is at D_{R_0}. That is, R_0 is an index to the quadruple which holds the root-node of our tree, wherever it may be in D. Thus R, considered as a vector, is defined as R[0:s], for some value s, and is thus indexed using 0-origin indexing. V, L, and B, on the other hand, are not accessed at zero, and hence, may be considered to be declared as V[1:s], L[1:s], and B[1:s]. We shall assume that R_0 is initially zero, indicating that no nodes exist.

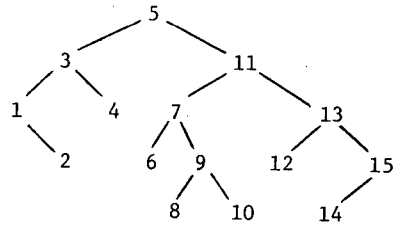
The tree stored in D is always organized such that for any node indexed by i, $V_{L_i} < V_i$ and $V_{R_i} > V_i$ whenever L_i and R_i are not zero, respectively.

Now, for our binary tree, consider each node $D_i = (V_i, L_i, R_i, B_i)$, and define h(i) as the height of the sub-tree with root node D_i. That is, h(i) is the number of nodes in a longest path in the sub-tree with root node i. Also h(0) = 0, since we understand the longest path of the null tree to be of length zero. Then we may define B_i for an arbitrary node D_i as follows:

$$B_i = h(R_i) - h(L_i) .$$

We shall say a tree is balanced if $|B_i| < 2$ for every node i in the tree. Clearly we should not demand more, since a number of nodes which is not a power of two less one cannot be organized so that every B_i is zero. A balanced tree thus approximates our notion of a tree where no excessively long or short paths exist. Moreover a stronger notion of balance would require excessive labor to maintain a balanced tree upon the addition or deletion of items.

Consider the following example:

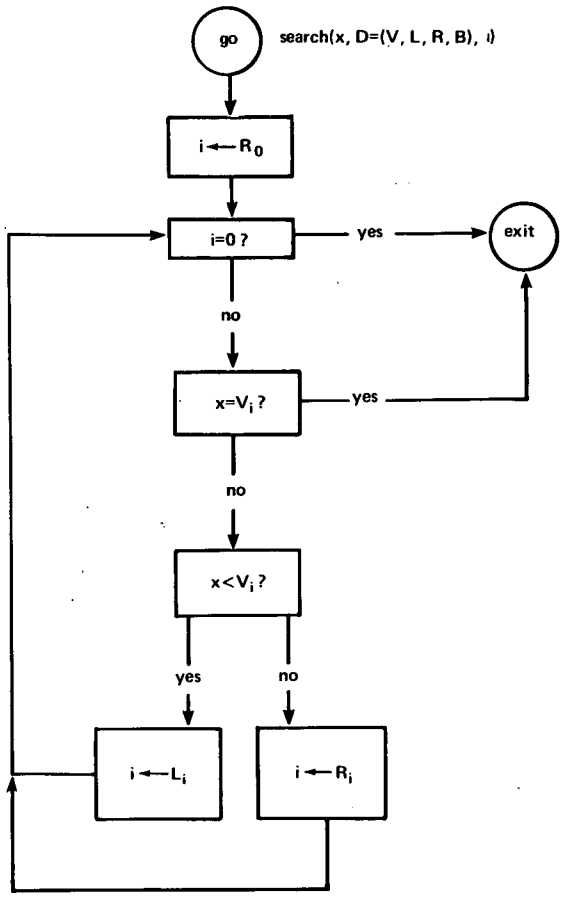


Here the integers represent the key values of the various nodes. This is seen to be a balanced tree by inspection. The above tree may actually be stored in D as follows:

	D			
	V	L	R	B
0			4	
1	10	0	0	0
2	1	0	6	1
3	3	2	12	-1
4	5	3	14	1
5	8	0	0	0
6	2	0	0	0
7	6	0	0	0
8	14	0	0	0
9	13	13	15	1
10	7	7	11	1
11	9	5	1	0
12	4	0	0	0
13	12	0	0	0
14	11	10	9	0
15	15	8	0	-1
f = 16	17	-	-	-
.				
.				
.				

3. The Search Procedure

The search routine for our defined tree is simple. It takes the argument x, which is a key-value to be retrieved in D (we will consistently confuse the key of an item with the item itself), and the argument D = (V, L, R, B). It returns the final argument i as the index in D of the first matching node, or as zero if there is no matching node. Thus we have:



4. The Storage Procedure

The add procedure for inserting a new node is more complex. It works in the following manner:

Given a new item x to be added, a search is done to discover what new leaf node position the new item is to occupy. This search defines a particular path from the root to that leaf node. Unlike the simple search given above, the path so defined must be remembered. A local push-down stack is a suitable device to save the successive nodes of such a path together with an indication of the direction from which they were exited.

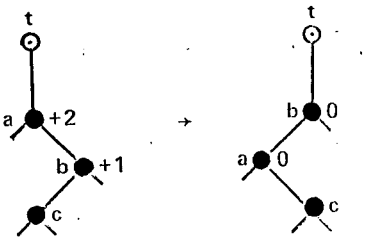
When x has been placed in D, the search path is retraced from bottom to top. At each previously-passed node, the balance at that node is recalculated, depending upon the direction of exit during the original traverse. Whenever a balance value indicates we have created an unbalanced sub-tree (i.e. whenever $|B_i|=2$), we shall immediately adjust the linkages in that sub-tree, including changing the root by changing a link which is "exterior" to the given sub-tree, so as to obtain a balanced tree with height equal to the previous height — before the addition occurred.

Note this can always be done because there is always some space or slack in an unbalanced tree. If every space were filled, we would have a perfect binary tree, but such a tree cannot be unbalanced.

Having balanced the first sub-tree which threatens the balance of the main tree, no further including tree will become unbalanced, nor will the various B-values be changed. Hence we are done. If during retracing we update all the B-values without needing to balance a sub-tree, then again we are finished. In particular, if we change a B-value to zero, then we may be assured we shall not need to balance any sub-tree, for no further B-values will change, once a zero is obtained. This is because if +1 (or -1) becomes a zero, then we have in fact only more perfectly balanced our tree.

4.1 The Balancing Algorithm

Now it remains only to describe the balancing algorithm. Whenever a B-value is +2 or -2, a right or left son respectively of that node exists and has a B-value of +1 or -1. Thus we have four cases. They will require two essentially different transformations. The basic idea is to "rotate" the tree to be balanced in the "direction" of its "small side." The simplest of the basic transformations is shown by example as:



The integers shown are the various balances. Here we have rotated the tree to the left. In terms of the actual representation we have:

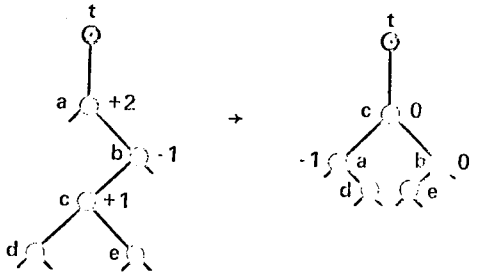
	V	L	R	B
t	V_t	a	R_t	B_t
a	V_a	L_a	b	2
b	V_b	c	R_b	1
c	V_c	L_c	R_c	B_c

→

	V	L	R	B
t	V_t	b	R_t	B_t
a	V_a	L_a	c	0
b	V_b	a	R_b	0
c	V_c	L_c	R_c	B_c

where we have assumed the sub-tree shown was reached from the left link of node D_t.

The second transformation arises when a simple rotation again produces an unbalanced tree. In such a case a deeper rotation will set things right. This case is shown in the following example.



Or, in terms of the actual representation, we have:

	V	L	R	B
t	V_t	a	R_t	B_t
a	V_a	L_a	b	2
b	V_b	c	R_b	-1
c	V_c	d	e	1
d	V_d	L_d	R_d	B_d
e	V_e	L_e	R_e	B_e

→

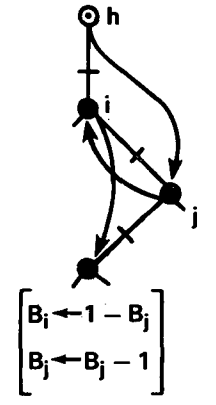
	V	L	R	B
t	V_t	c	R_t	B_t
a	V_a	L_a	d	-1
b	V_b	e	R_b	0
c	V_c	a	b	0
d	V_d	L_d	R_d	B_d
e	V_e	L_e	R_e	B_e

Here again, we have assumed that the left link of the "master node," D_t, pointed to node D_a.

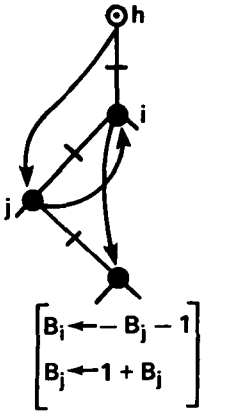
Note that the transformations just given remain valid when the node D_c in the first case, and D_d and/or D_e in the second, do not exist.

Now some contemplation will show that only these two basic situations together with their reflected images can ever occur. Actually the case where we have a node with a balance value of ±2 and a son or sons with a balance value of 0 will arise later and is covered below. The only further complication is the recomputation of B-values in all cases. The entire situation is shown below. Here the bracketed expressions show the required B-value recomputations in each case. It can easily be seen that they are, in fact, correct. The links shown with arrow heads are "new." "Old" links which are changed are "cut" with a slash. Unlabeled nodes need not exist in which case the referent links are zero. The link coming from the master node (shown as ⊙), which is the parent node of the unbalanced sub-tree, may be in R_0 or may be a left or right link of some actual node. Its logical position remains unchanged.

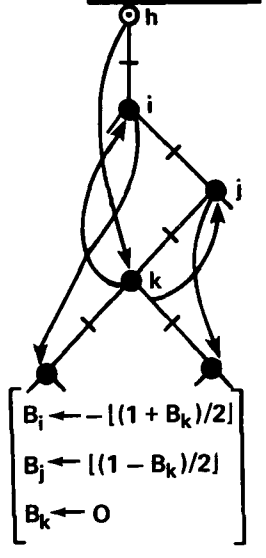
Case 1: $B_i = 2$, $B_j \geq 0$.



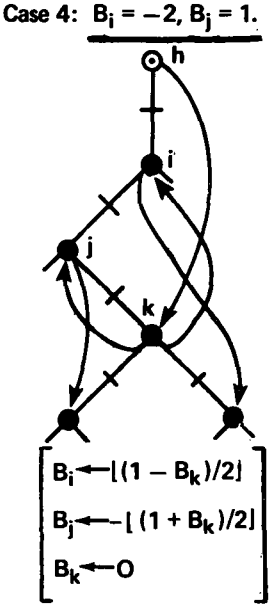
Case 2: $B_i = -2$, $B_j \leq 0$.



Case 3: $B_i = 2$, $B_j = -1$.



Case 4: $B_i = -2$, $B_j = 1$.



The transformations given above must be used to maintain the balance of our tree when updating, i.e. adding or deleting an item. Thus we give below a routine called balance to be used as a subroutine for invoking the various balance transformations. Its arguments are the area D = (V, L, R, B); a set of indices to nodes in D, namely h, i, j, and k; and a switch, d, which is to be +1 or -1. h is the index of the master node which has the node indexed by i as a son. The i-node is the left son of the h-node if d is -1, and is a right son of the h-node if d = +1. The balance, B_i, at the i-node is assumed to be +2 or -2. j is the index of the right son of the i-node if B_i = +2 and is the index of the left son of the i-node if B_i = -2. Finally if $B_iB_j < 0$ then k is the index of the left-son of the j-node if B_i = +2, and is the index of the right son of the j-node if B_i = -2. If $B_iB_j \geq 0$, then k is not used and may be an arbitrary value.

It will be seen that the routine given below merely encodes the transformations given in the diagrams above.

4.2 Various Item Addition Algorithms

Now we shall consider several versions of the add algorithm used to add an element to our tree. The required arguments are x, the element to be added; the area D; and the free-space pointer f.

In the first program a local stack, S, is used to save the search trail. In the second version a compactification of code is achieved by a suitable "renaming" imposed on D. In the third version, the search trail is kept in D itself and the appropriate pointers are restored as the backscan occurs.

Recall that R_0 is assumed to be zero initially, and that f is suitably initialized.

The first add program we consider is called basic-add. The balance program, balance, is invoked as a subroutine.

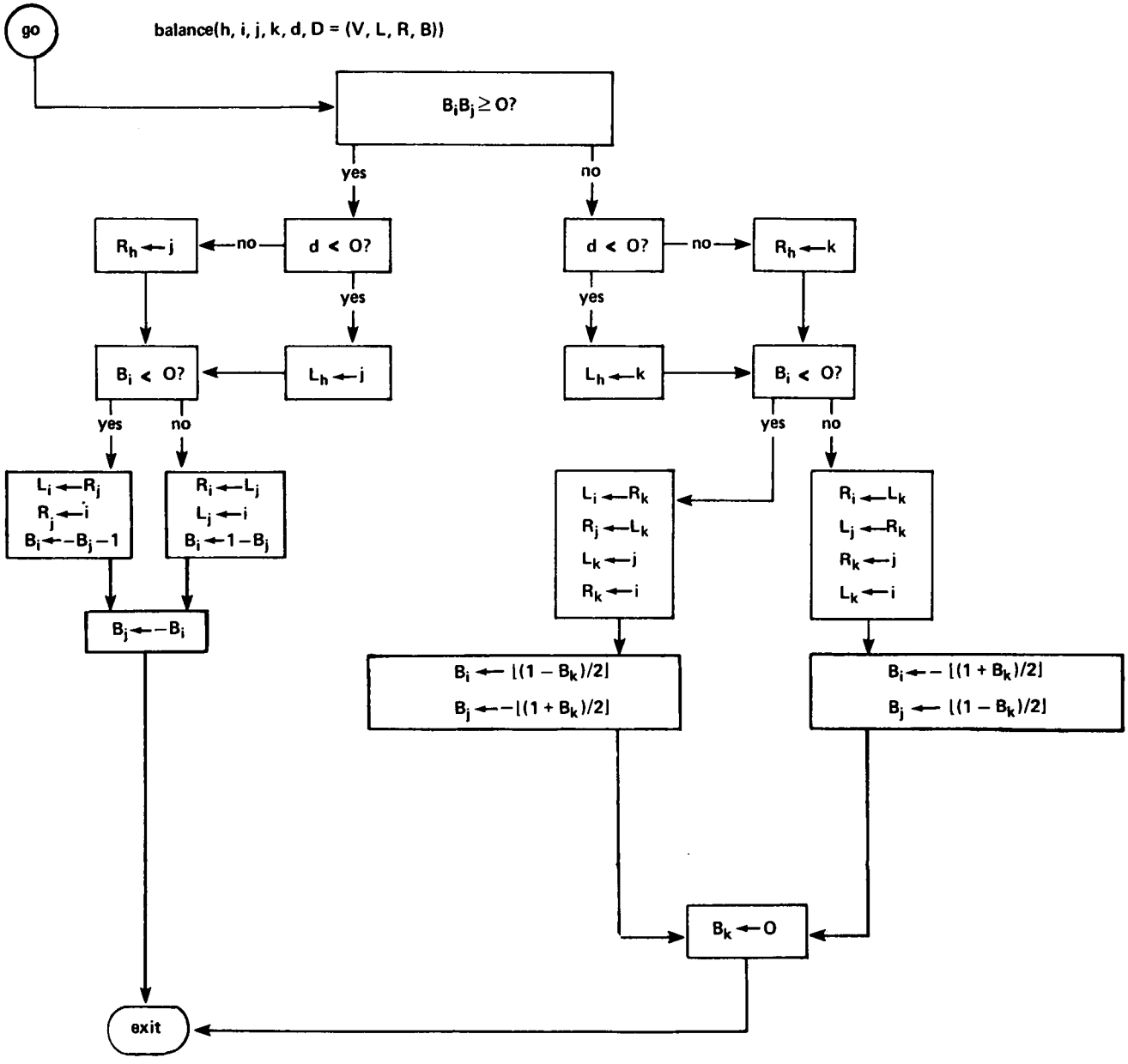
The basic-add program can be expressed more tersely.

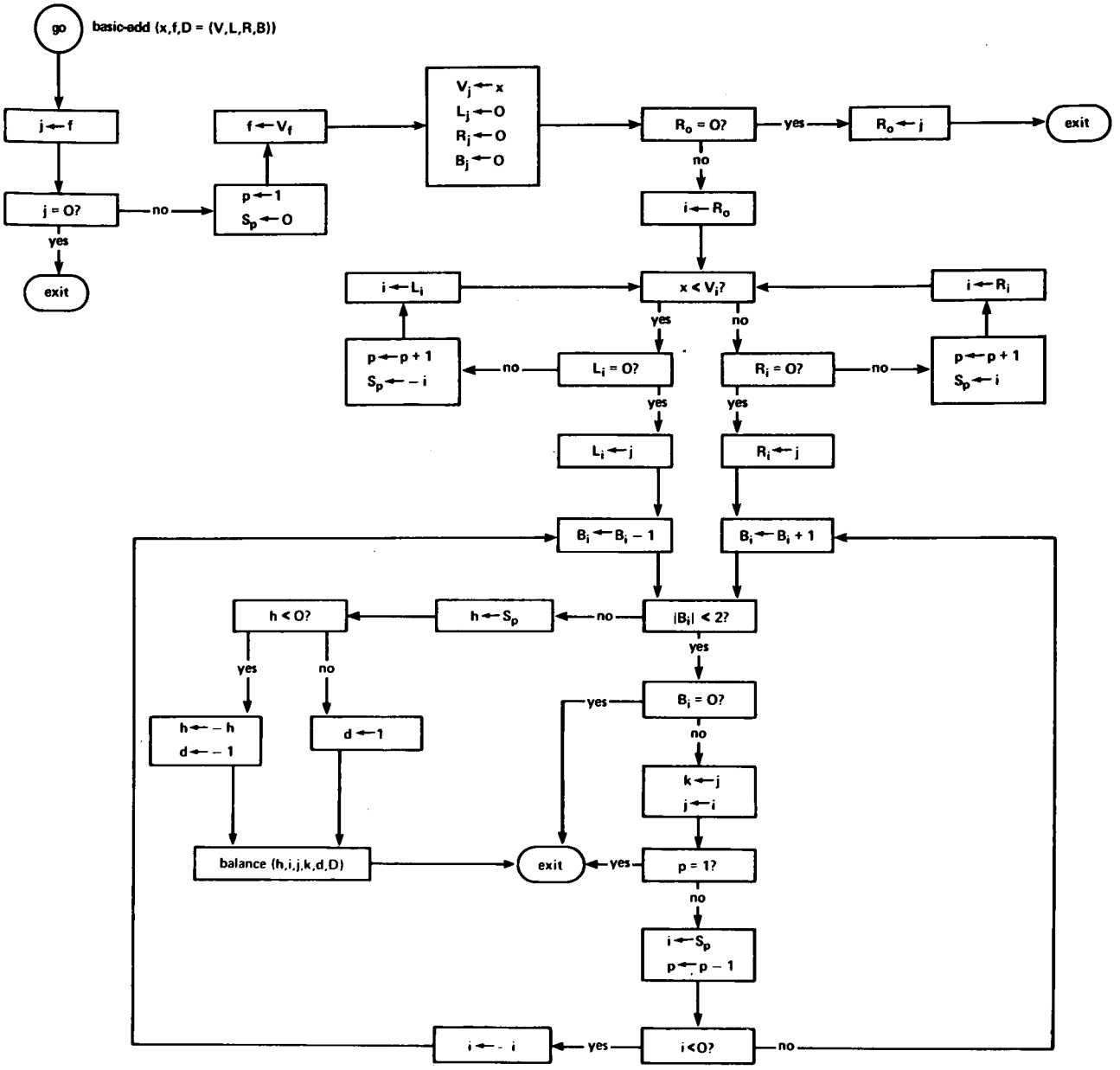
The formulation of the following algorithm, called new-basic-add, indicates some of the economies due to symmetry which one would consider in actually coding a set of programs for use in balanced tree storage and retrieval.

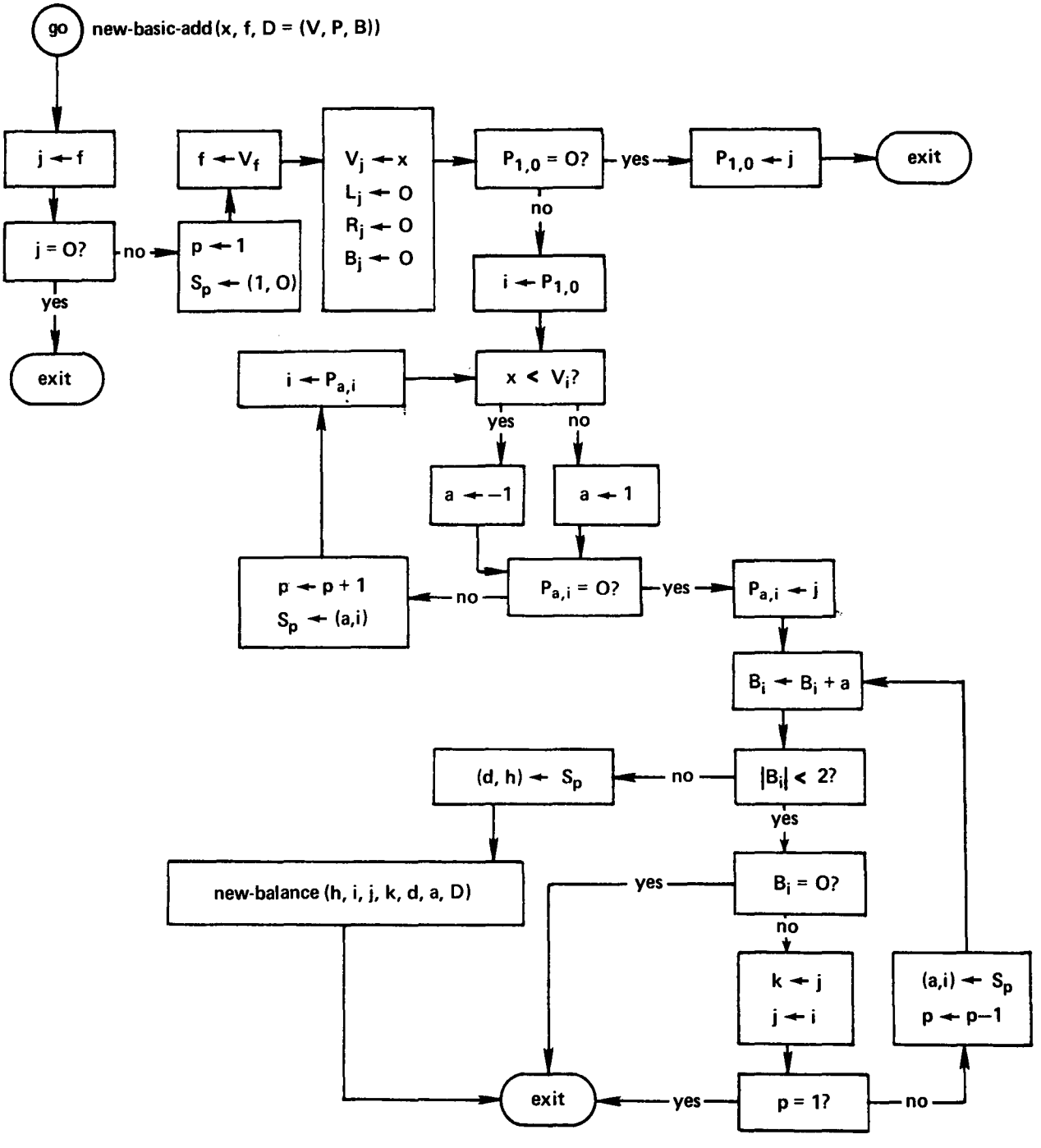
The balance routine must be modified to accept another parameter which determines which reflection of the appropriate balancing transformation is to be used. We call this modified balance routine new-balance, and it is given below following its calling program, new-basic-add. Also the stack S is used here to hold a "packed" entry or vector in each position. The underlying detailed logic should be clear.

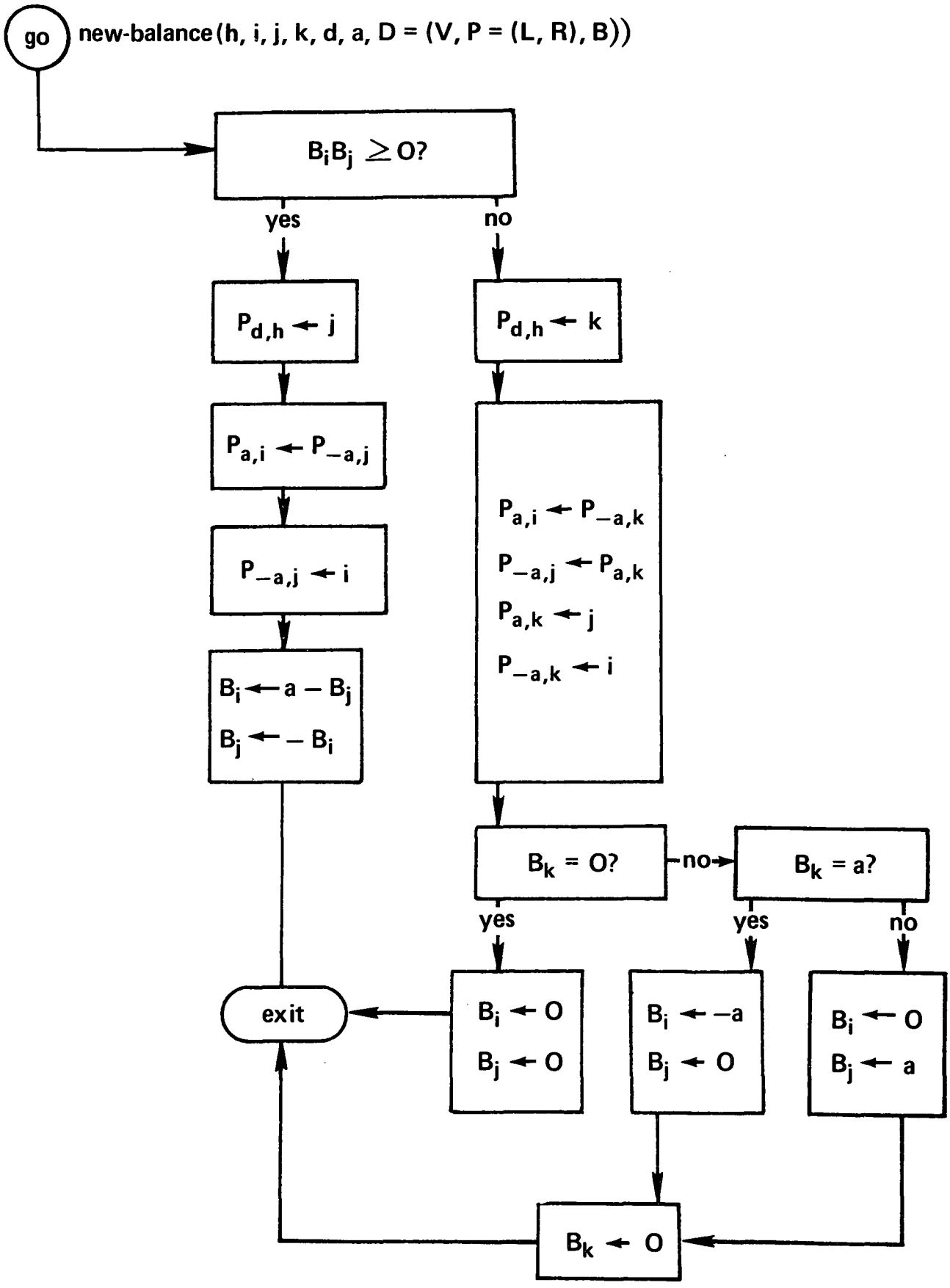
Finally, the add routine can also be given without the use of an auxiliary stack. Rather we may reverse pointers on the way "down" and re-reverse them on the way back "up" when the balance values, B_i, are being updated. The sign of a link will indicate its status and the position (L_i or R_i) of a negative link indicates whether the son of interest was a left son or right son. The "back-chain" thus defined terminates when a self-referent link is found. This program is given below as the routine called add. The balance program given originally is used as a subroutine.

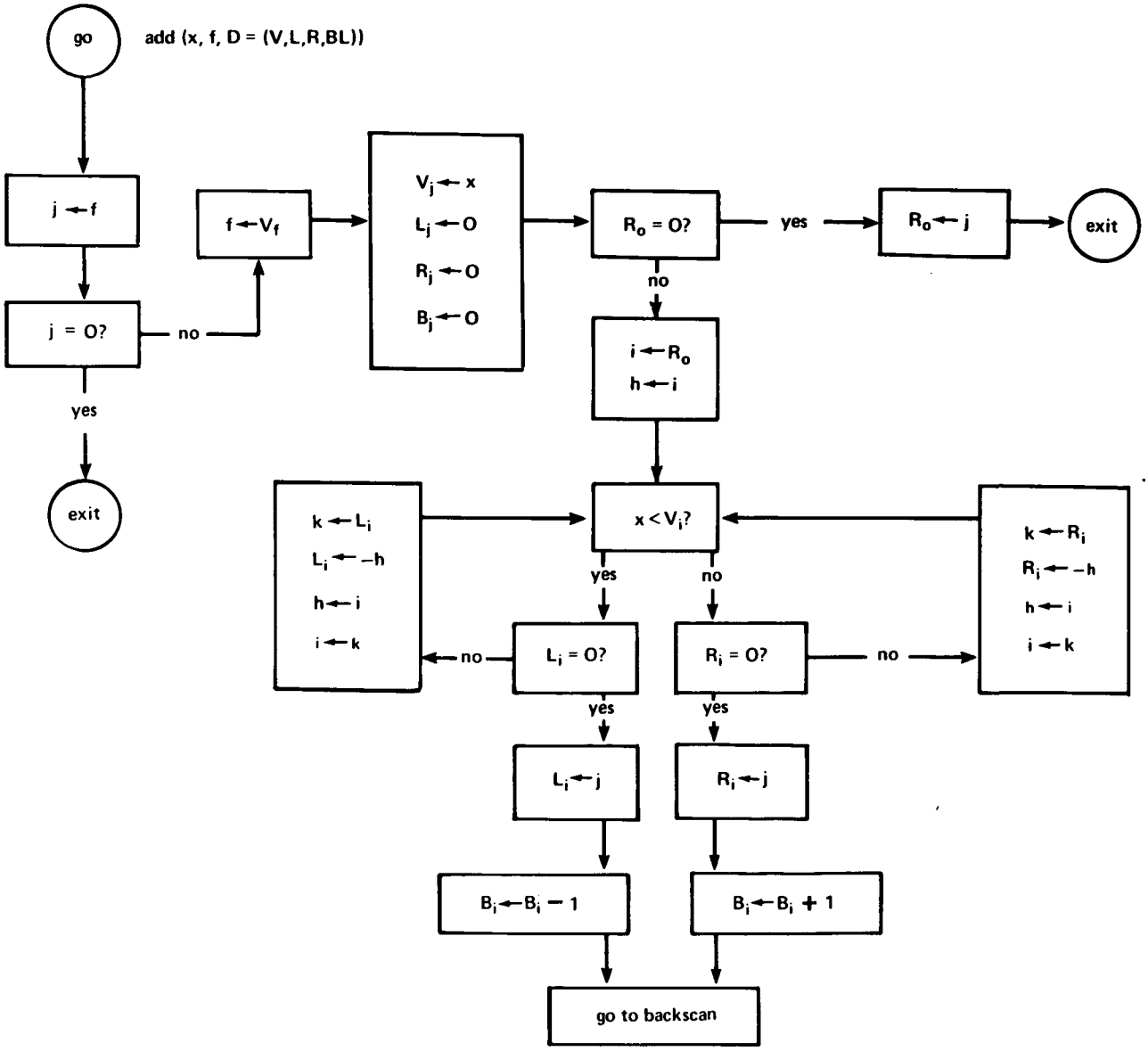
We now give the balance routine and the various addition routines.

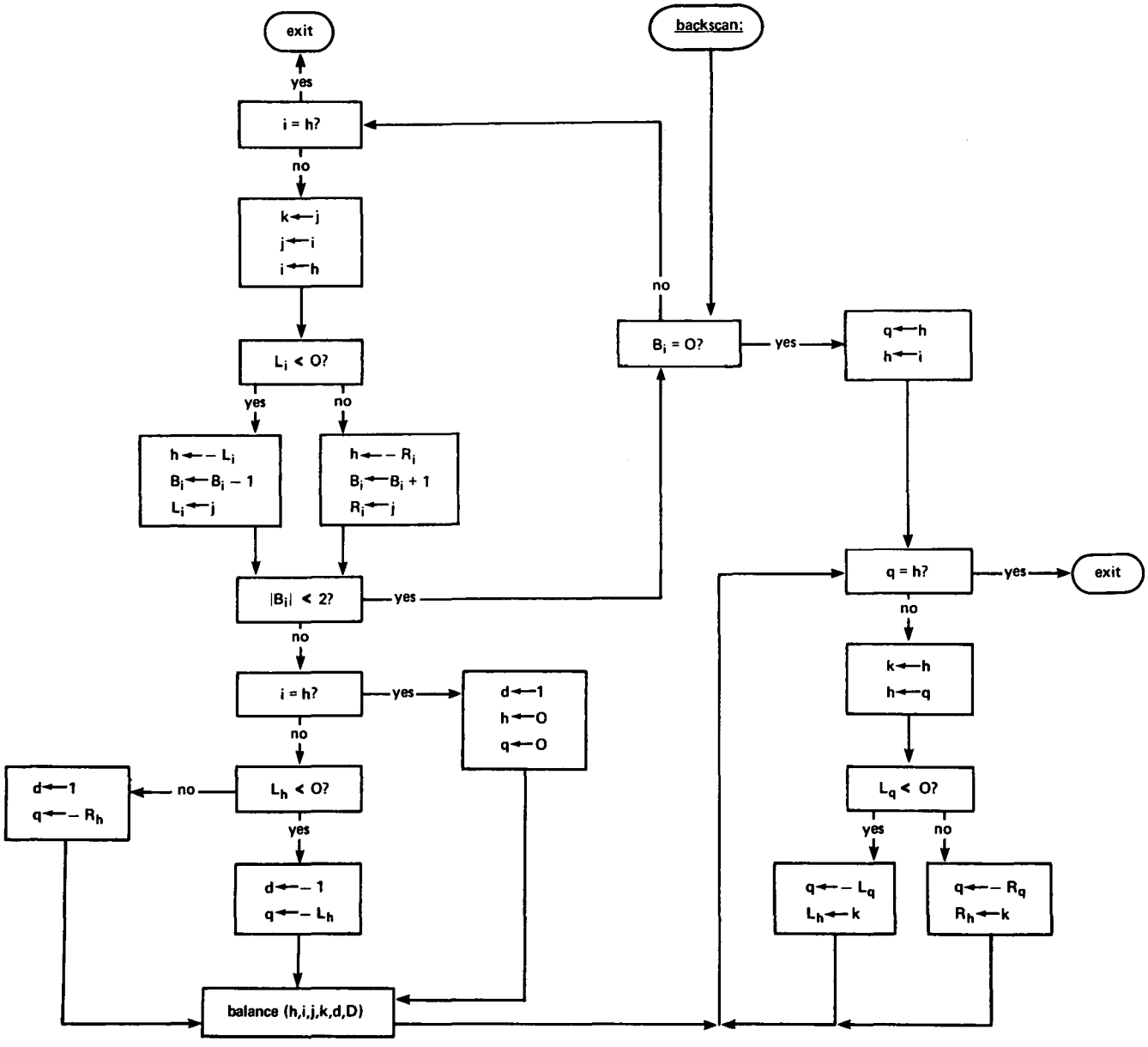

go
balance(h, i, j, k, d, D = (V, L, R, B))
$B_iB_j \geq 0$?
yes
no
d < 0?
$R_h \leftarrow j$
no
yes
B_i < 0?
$L_h \leftarrow j$
yes
no
$L_i \leftarrow R_j$
$R_j \leftarrow i$
$B_i \leftarrow -B_j - 1$
$R_i \leftarrow L_j$
$L_j \leftarrow i$
$B_i \leftarrow -1 - B_j$
$B_j \leftarrow -B_i$
d < 0?
no
$R_h \leftarrow k$
yes
$L_h \leftarrow k$
B_i < 0?
yes
no
$L_i \leftarrow R_k$
$R_j \leftarrow L_k$
$L_k \leftarrow j$
$R_k \leftarrow i$
$R_i \leftarrow L_k$
$L_j \leftarrow R_k$
$R_k \leftarrow j$
$L_k \leftarrow i$
$B_i \leftarrow \lfloor (1 - B_k)/2 \rfloor$
$B_j \leftarrow -\lfloor (1 + B_k)/2 \rfloor$
$B_i \leftarrow -\lfloor (1 + B_k)/2 \rfloor$
$B_j \leftarrow \lfloor (1 - B_k)/2 \rfloor$
$B_k \leftarrow 0$
exit


go
basic-add (x,f,D = (V,L,R,B))
j ← f
j = 0?
no
yes
exit
p ← 1
S_p ← 0
f ← V_f
V_j ← x
L_j ← 0
R_j ← 0
B_j ← 0
R_o = 0?
yes
R_o ← j
exit
no
i ← R_o
i ← L_i
x < V_i?
i ← R_i
yes
no
p ← p + 1
S_p ← − i
no
L_i = 0?
R_i = 0?
no
p ← p + 1
S_p ← i
yes
yes
L_i ← j
R_i ← j
B_i ← B_i − 1
B_i ← B_i + 1
h < 0?
h ← S_p
no
|B_i| < 2?
yes
yes
no
B_i = 0?
yes
h ← − h
d ← − 1
d ← 1
no
k ← j
j ← i
balance (h,i,j,k,d,D)
exit
yes
p = 1?
no
i ← S_p
p ← p − 1
i ← − i
yes
i<0?
no


go
new-basic-add (x, f, D = (V, P, B))
j ← f
j = O?
no
yes
exit
p ← 1
S_p ← (1, O)
f ← V_f
V_j ← x
L_j ← O
R_j ← O
B_j ← O
$P_{1,0}$ = O?
yes
$P_{1,0}$ ← j
exit
no
i ← $P_{1,0}$
i ← $P_{a,i}$
x < V_i?
yes
no
a ← −1
a ← 1
p ← p + 1
S_p ← (a,i)
no
$P_{a,i}$ = O?
yes
$P_{a,i}$ ← j
B_i ← B_i + a
(d, h) ← S_p
no
$|B_i|$ < 2?
yes
new-balance (h, i, j, k, d, a, D)
yes
B_i = O?
no
k ← j
j ← i
(a,i) ← S_p
p ← p−1
exit
yes
p = 1?
no


go
new-balance(h, i, j, k, d, a, D = (V, P = (L, R), B))
$B_iB_j \geq 0$?
yes
no
$P_{d,h} \leftarrow j$
$P_{d,h} \leftarrow k$
$P_{a,i} \leftarrow P_{-a,j}$
$P_{-a,j} \leftarrow i$
$B_i \leftarrow a - B_j$
$B_j \leftarrow - B_i$
$P_{a,i} \leftarrow P_{-a,k}$
$P_{-a,j} \leftarrow P_{a,k}$
$P_{a,k} \leftarrow j$
$P_{-a,k} \leftarrow i$
$B_k = 0$?
no
$B_k = a$?
yes
yes
no
exit
$B_i \leftarrow 0$
$B_j \leftarrow 0$
$B_i \leftarrow -a$
$B_j \leftarrow 0$
$B_i \leftarrow 0$
$B_j \leftarrow a$
$B_k \leftarrow 0$


go
add (x, f, D = (V,L,R,BL))
j ← f
j = 0?
no
yes
exit
f ← V_f
V_j ← x
L_j ← 0
R_j ← 0
B_j ← 0
R_0 = 0?
yes
R_0 ← j
exit
no
i ← R_0
h ← i
x < V_i?
yes
no
k ← L_i
L_i ← −h
h ← i
i ← k
no
L_i = 0?
yes
L_i ← j
B_i ← B_i − 1
k ← R_i
R_i ← −h
h ← i
i ← k
no
R_i = 0?
yes
R_i ← j
B_i ← B_i + 1
go to backscan


exit
yes
i = h?
no
k←j
j←i
i←h
Li < O?
yes
no
h←– Li
Bi←Bi – 1
Li←j
h←– Ri
Bi←Bi + 1
Ri←j
|Bi| < 2?
yes
no
i = h?
yes
d←1
h←O
q←O
no
d←1
q←– Rh
no
Lh < O?
yes
d←– 1
q←– Lh
balance (h,i,j,k,d,D)
backscan:
no
Bi = O?
yes
q←h
h←i
q = h?
yes
exit
no
k←h
h←q
Lq < O?
yes
no
q←– Lq
Lh←k
q←– Rq
Rh←k

5. The Deletion Procedure

The deletion algorithm for our balanced tree storage scheme is somewhat difficult when considered in detail although it is very similar in spirit to the add procedures discussed above.

The deletion process proceeds by searching for the node which is to be deleted, given its key value. The path taken during this search must be saved for possible later retracing. If no matching node is found, we are done. Otherwise, the node to be deleted is removed from our tree by suitably relinking various surrounding nodes, and the space in D used for the deleted node is returned to the free-space list.

The required relinking may result in unbalancing our tree at a node of the disturbed region. Thus this situation must be tested for and, if present, may be corrected by an application of the balance routine given earlier. Unlike the add procedure, the deletion procedure may perform several balance operations; since, after relinking, we must retrace our steps back to the root, adjusting the balance values of each node of the back-trail as we go. Whenever the sub-tree defined by the current node of the back-trail is unbalanced, we must pause and apply the balancing algorithm before continuing. It is possible that a balancing operation will be required for every node in the back-trail, although this is not normally the case, for when a balance value becomes +1 or -1, no further changes will occur.

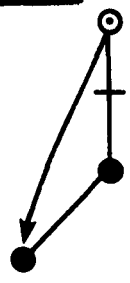
5.1 Deletion Relinking

We may now discuss in more detail the relinking process required to remove a node from our tree, whereupon we may further consider the "rewinding and rebalancing" logic.

It will suffice to give several general diagrams showing the transformations required. It can be seen that the situations shown below are exhaustive. As before, in each of the diagrams below, ? represents the master node (in this case the node which is the father of the node to be deleted). If the root node is being deleted, the master node consists only of the R_0 "entry" link. Thus, in each case, the node labeled j is to be removed. Also, we understand that in each diagram an unlabeled node need not exist, in which case the referent link shown is zero. The transformations given remain valid under this interpretation. Moreover, as before, old links are shown by simple lines while new links have arrow heads shown. Finally, we note that it is easy to justify the balance value recomputations shown. The balance values of nodes above node i may not be correct, but this is taken care of in the course of the deletion process.

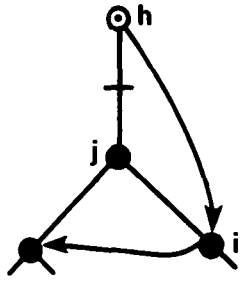
In general, the transformations given merely "replace" the deleted node with a more-or-less close neighbor such that the "little-to-the-left, big-to-the-right" structure of our tree is preserved. Thus, we have the following situations. Note that case 3 is just a special case of case 4, wherein t=i.

Case 1: $R_j = 0$.



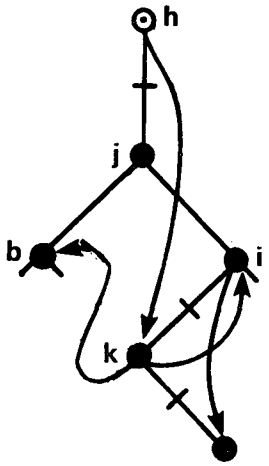
$[B_i \leftarrow B_i + 1$ if j is the left son of i, otherwise, $B_i \leftarrow B_i - 1.]$

Case 2: $L_i = 0$.



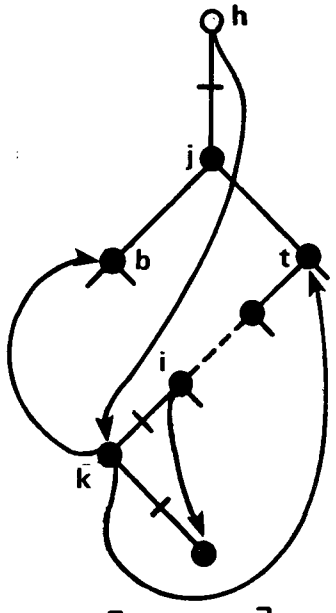
$[B_i \leftarrow B_j - 1]$

Case 3: $L_i = k$.



$$\begin{bmatrix} B_i \leftarrow B_i + 1 \\ B_k \leftarrow B_j \end{bmatrix}$$

Case 4: $L_t \neq L_i = k$.



$$\begin{bmatrix} B_i \leftarrow B_i + 1 \\ B_k \leftarrow B_j \end{bmatrix}$$

5.2 Various Deletion Algorithms

After the node to be deleted has been removed by means of the appropriate relinking transformation, we then consider the back-trail from the node labled i back to the root of our tree.

It will be seen that after relinking, the balance at node i is correct (and possibly ± 2) and the balance at each node which is a parent of node i may be incorrect by +1 or -1. Thus we first check the balance at node i and balance the subtree with root node i (which we now call the current sub-tree) if required. This may leave the current sub-tree with a height which is one less than the corresponding sub-tree in the same position (this is, with root node i) before we began the deletion process. This is the case if the current sub-tree before any required rebalancing is done, has $B_i=0$, or $B_i=2$ and $B_{R_i}=\pm 1$, or $B_i=-2$ and $B_{L_i}=\pm 1$. In these circumstances, after any necessary rebalancing, we see that we have "shortened the long side" of the current sub-tree and hence decreased its total height. Otherwise, we have $B_i=\pm 1$, or $B_i=2$ and $B_{R_i}=0$, or $B_i=-2$ and $B_{L_i}=0$, and after any necessary rebalancing the resulting current sub-tree will not have decreased in height. These two situations can be distinguished by the fact that <u>after</u> any necessary rebalancing is done, we have the balance value of the current sub-tree equal to zero if the height has decreased and ± 1 otherwise. Thus in case we obtain a balance value of zero, the resulting sub-tree is shorter by one than originally and hence, if the balance at its parent node is dependent upon the current sub-tree's previous height, we may find it necessary to rebalance the sub-tree whose root is the just-higher node in our back-trail. In particular, this additional rebalancing will be required only when our sub-tree (with root node i, unless balanced) has grown shorter in height by one <u>and</u> also, either its parent node has a balance value of +1 and our current sub-tree is to the left of the parent node, or its parent node has a balance value of -1 and our current sub-tree is to the right of the parent node.

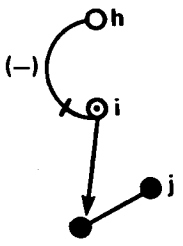
We may now see how we must proceed up the back-trail. If we enter a node from its left son, we increase the balance at the node by one, while if we enter a node from its right son, we decrease the balance at the node by one. If we thereby obtain a new balance value of +1 or -1, we are done and no further balance adjustments are required. This is because the height of our current sub-tree has <u>not</u> decreased. If, on the other hand, rebalancing is required, it is invoked. Then if the resulting sub-tree has a balance value of +1 or -1, we are done, since again the height of the current sub-tree has not changed. Otherwise, we must continue to the next node in the back-trail.

The process described above is given precisely in the routine below called basic-delete. The arguments are x, the key of the item to be deleted; f, the free-space pointer; and D, our basic tree area. The local stack, S, is used to save the back-trail as in the basic-add routine given earlier. Finally, the balance routine given earlier is used below.

We also give below the deletion algorithm which corresponds to the add procedure given earlier. This routine, called delete, takes the same arguments as basic-delete, but no local stack, S, is needed. Rather, the back-trail is kept by temporarily reversing pointers during the initial search.

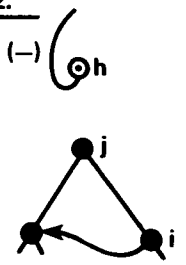
In this routine the logic of the relinking transformations given earlier is changed to correctly remove a node from our tree with the back-trail being maintained. The essential changes are shown in the following set of diagrams, where a link labeled (-) is understood to be part of our back-trail; otherwise, all our conventions are as before. Again case 3 is just a special case of case 4.

Case 1.



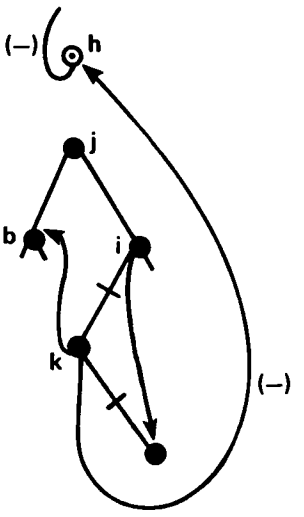
$$\begin{bmatrix} B_i \leftarrow B_i + 1, \text{ if } j \text{ is the left son of } i. \\ B_i \leftarrow B_i - 1, \text{ otherwise.} \end{bmatrix}$$

Case 2.



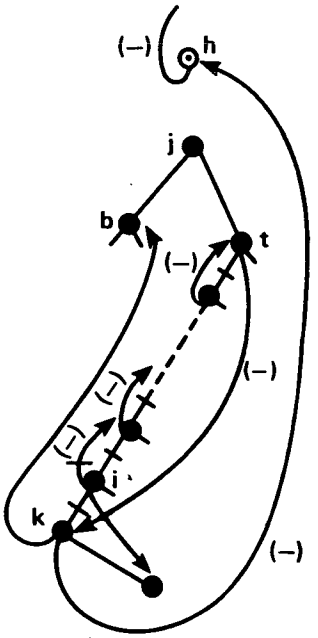
$$\left[B_i \leftarrow B_j - 1\right]$$

Case 3.



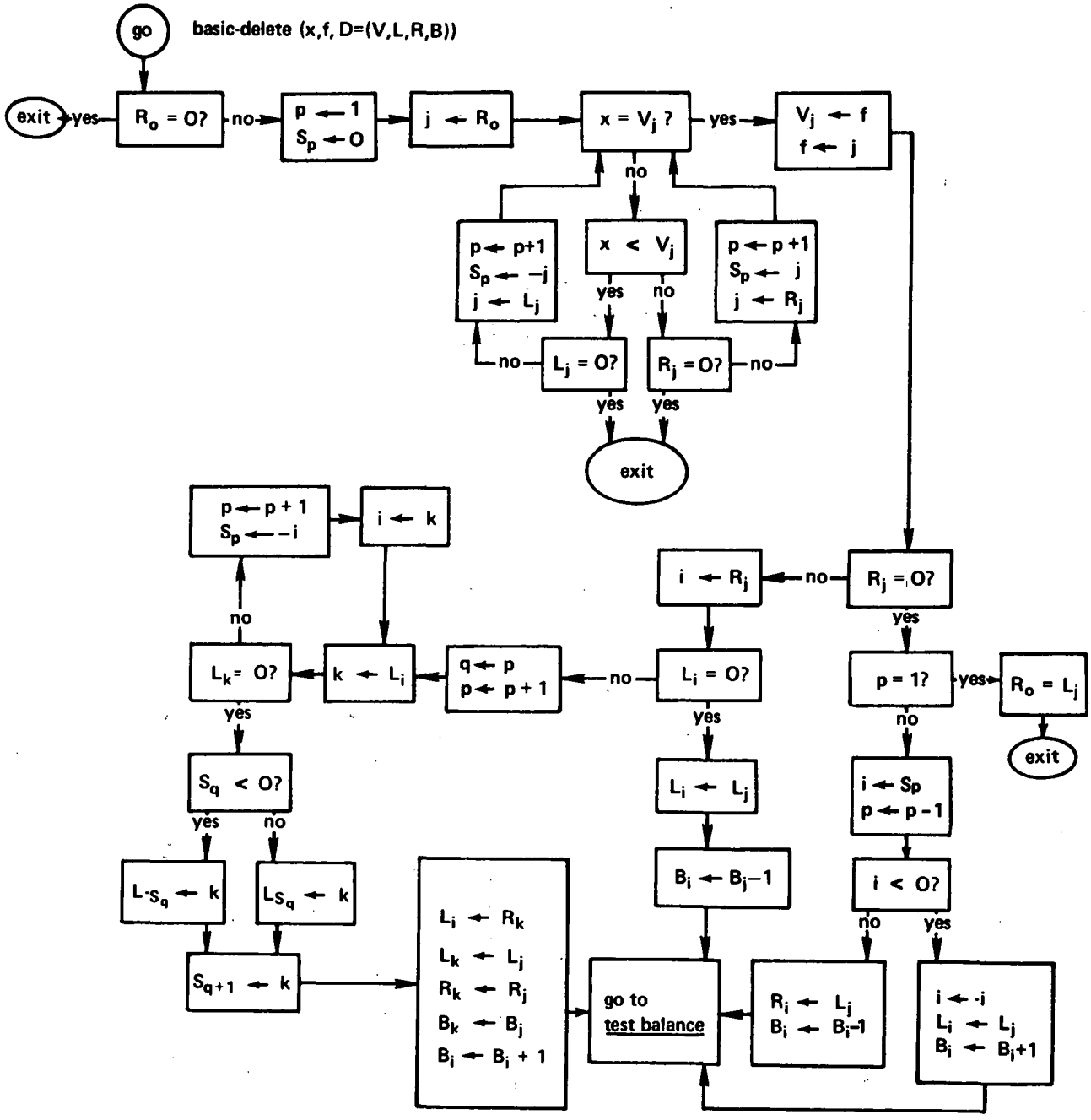
$$\begin{bmatrix} B_i \leftarrow B_i + 1 \\ B_k \leftarrow B_j \end{bmatrix}$$

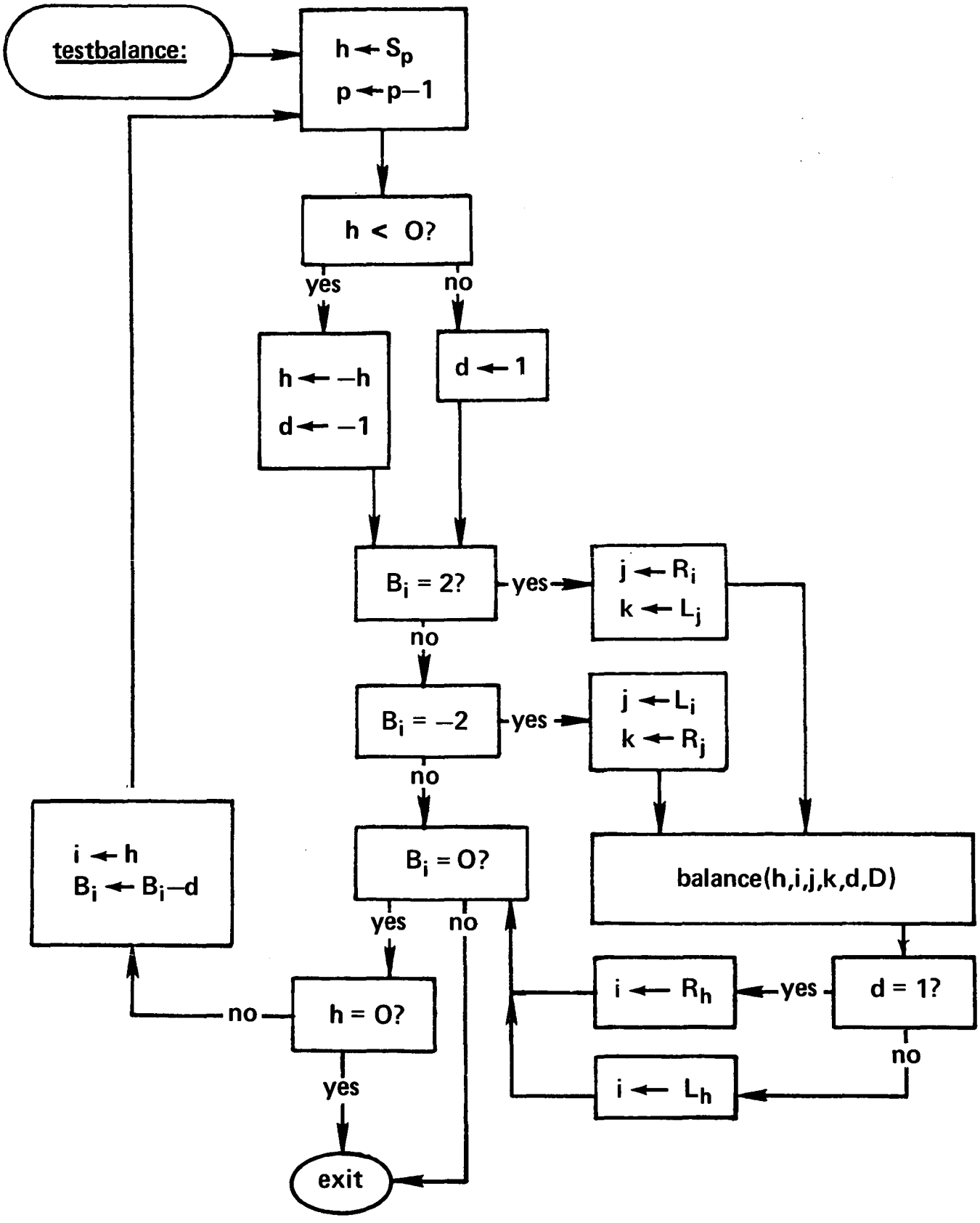
Case 4.

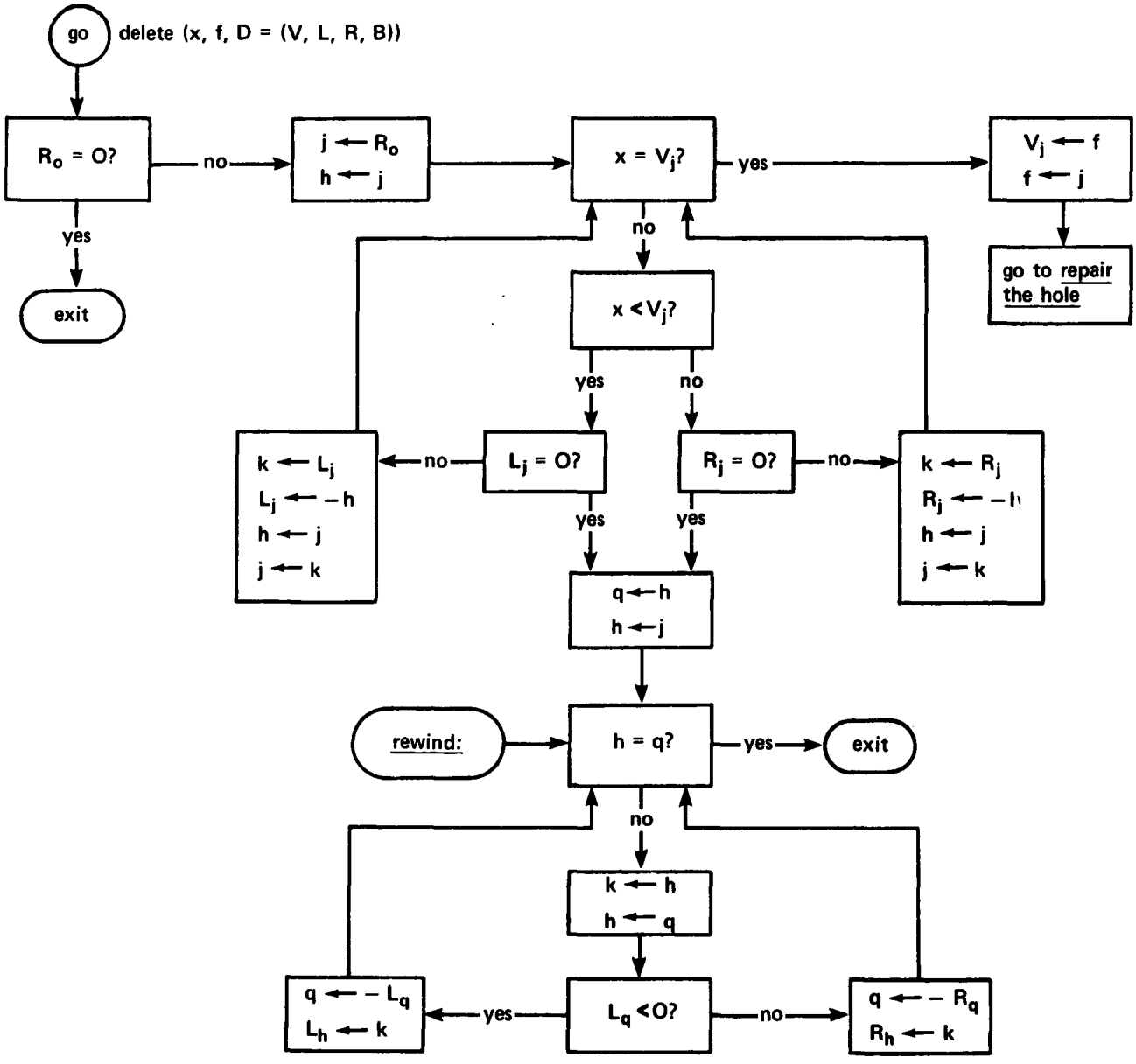


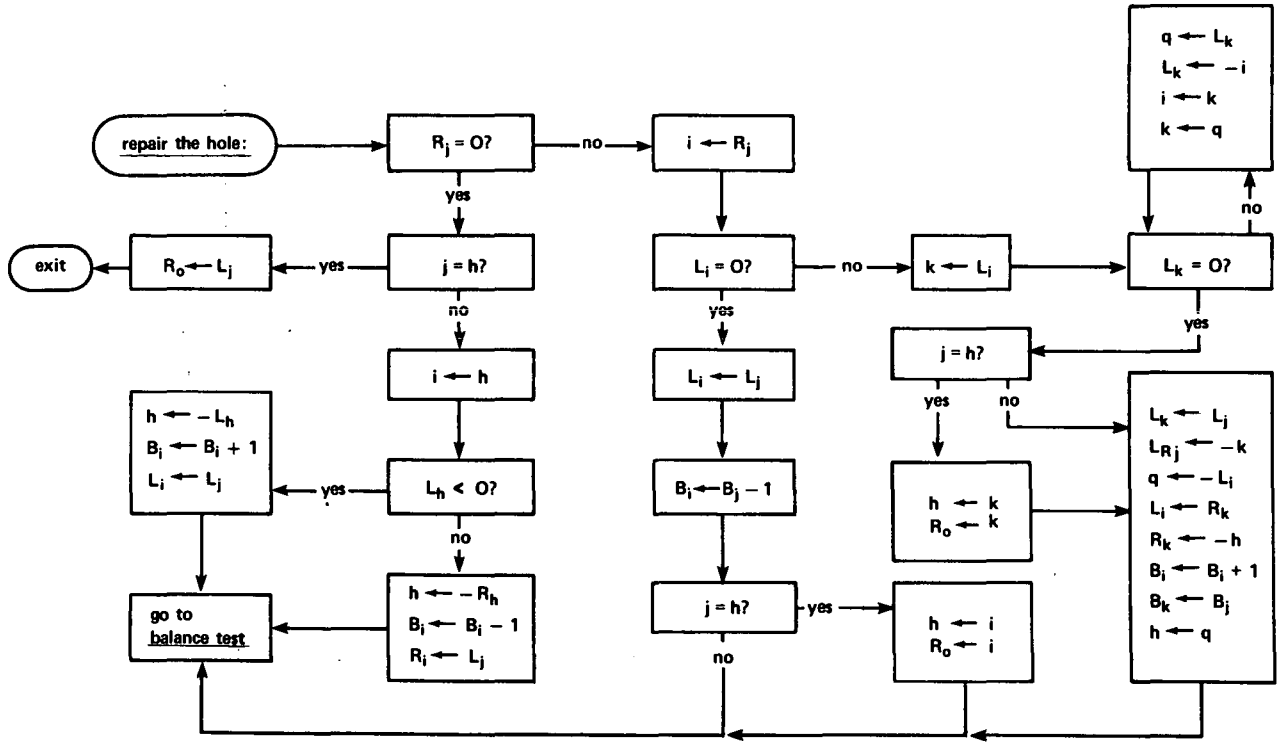
$$\begin{bmatrix} B_i \leftarrow B_i + 1 \\ B_k \leftarrow B_j \end{bmatrix}$$

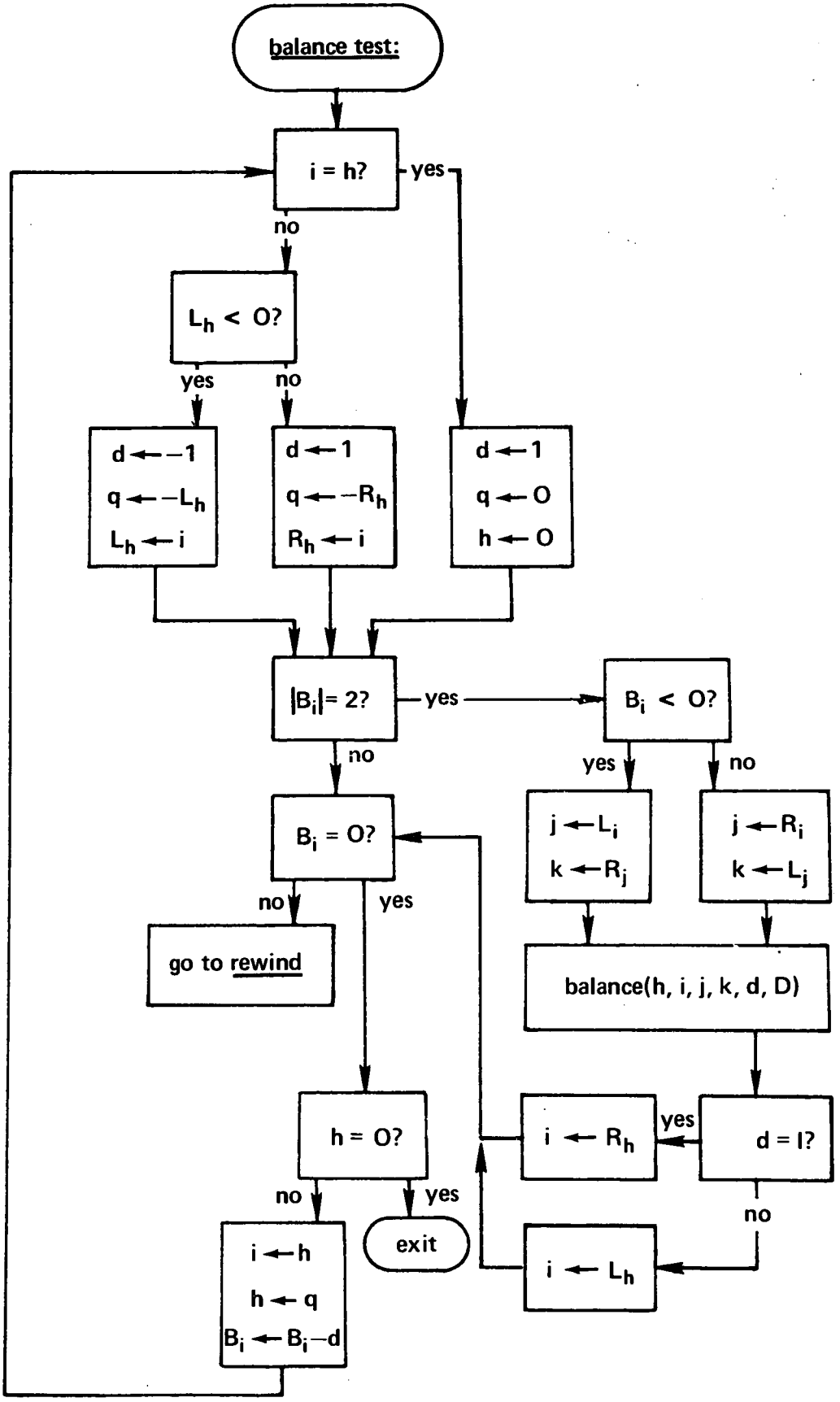
We now give the various deletion routines. basic-delete is followed by delete which uses the transformations given just above.


go
basic-delete (x,f, D=(V,L,R,B))
exit
yes
R_o = O?
no
p ← 1
S_p ← O
j ← R_o
x = V_j ?
yes
V_j ← f
f ← j
no
p ← p+1
S_p ← −j
j ← L_j
x < V_j
yes
no
p ← p+1
S_p ← j
j ← R_j
no
L_j = O?
R_j = O?
no
yes
yes
exit
p ← p + 1
S_p ← −i
i ← k
i ← R_j
no
R_j = O?
yes
no
L_k = O?
k ← L_i
q ← p
p ← p + 1
no
L_i = O?
p = 1?
yes
R_o = L_j
exit
yes
yes
no
S_q < O?
L_i ← L_j
i ← Sp
p ← p - 1
yes
no
B_i ← B_j−1
i < O?
L_{-S_q} ← k
L_{S_q} ← k
L_i ← R_k
L_k ← L_j
R_k ← R_j
B_k ← B_j
B_i ← B_i + 1
S_{q+1} ← k
go to
test balance
R_i ← L_j
B_i ← B_i−1
i ← -i
L_i ← L_j
B_i ← B_i+1
no
yes


testbalance:
h ← Sp
p ← p−1
h < O?
yes
no
h ← −h
d ← −1
d ← 1
Bi = 2?
yes
j ← Ri
k ← Lj
no
Bi = −2
yes
j ← Li
k ← Rj
no
i ← h
Bi ← Bi−d
Bi = O?
balance(h,i,j,k,d,D)
yes
no
i ← Rh
yes
d = 1?
no
h = O?
no
i ← Lh
yes
exit


go
delete (x, f, D = (V, L, R, B))
R_o = O?
no
yes
exit
j ← R_o
h ← j
x = V_j?
yes
no
V_j ← f
f ← j
go to repair the hole
x < V_j?
yes
no
L_j = O?
no
k ← L_j
L_j ← −h
h ← j
j ← k
R_j = O?
no
k ← R_j
R_j ← −h
h ← j
j ← k
yes
yes
q ← h
h ← j
rewind:
h = q?
yes
exit
no
k ← h
h ← q
L_q < O?
yes
q ← − L_q
L_h ← k
no
q ← − R_q
R_h ← k


repair the hole:
$R_j = O$?
no
$i \leftarrow R_j$
yes
$q \leftarrow L_k$
$L_k \leftarrow -i$
$i \leftarrow k$
$k \leftarrow q$
no
exit
$R_o \leftarrow L_j$
yes
$j = h$?
$L_i = O$?
no
$k \leftarrow L_i$
$L_k = O$?
yes
no
yes
$i \leftarrow h$
$L_i \leftarrow L_j$
$j = h$?
yes
no
$h \leftarrow -L_h$
$B_i \leftarrow B_i + 1$
$L_i \leftarrow L_j$
yes
$L_h < O$?
$B_i \leftarrow B_j - 1$
$h \leftarrow k$
$R_o \leftarrow k$
$L_k \leftarrow L_j$
$L_{Rj} \leftarrow -k$
$q \leftarrow -L_i$
$L_i \leftarrow R_k$
$R_k \leftarrow -h$
$B_i \leftarrow B_i + 1$
$B_k \leftarrow B_j$
$h \leftarrow q$
no
go to
balance test
$h \leftarrow -R_h$
$B_i \leftarrow B_i - 1$
$R_i \leftarrow L_j$
$j = h$?
yes
$h \leftarrow i$
$R_o \leftarrow i$
no


balance test:
i = h?
yes
no
L_h < 0?
yes
no
d ← −1
q ← $-L_h$
L_h ← i
d ← 1
q ← $-R_h$
R_h ← i
d ← 1
q ← 0
h ← 0
$|B_i|$ = 2?
yes
B_i < 0?
yes
no
no
B_i = 0?
j ← L_i
k ← R_j
j ← R_i
k ← L_j
no
yes
go to rewind
balance(h, i, j, k, d, D)
h = 0?
i ← R_h
yes
d = l?
no
yes
no
exit
i ← L_h
i ← h
h ← q
B_i ← B_i − d

6. A Cost Analysis

Here we obtain an upper bound for C, the number of 3-way comparisons required to search for an item and retrieve it (if present) from a balanced tree of n nodes. Given a balanced tree of items and a key, each comparison either matches the key being searched for or allows us to make the next comparison at the next higher level in our tree. Thus, the desired upper bound is given as the maximum number of levels which can be found in some balanced tree of n nodes.

We begin by determining the minimum number of nodes, T_k, which may occur in a balanced tree with k levels.

Clearly, we have $T_0 = 0$ and $T_1 = 1$. Now we may see that $T_{k+1} = T_k + T_{k-1} + 1$ for $k \geq 1$, since we schematically have

$$\overline{T}_{k+1} = \quad \overline{T}_k \quad \overline{T}_{k-1} \qquad (1)$$

where here we understand that $\overline{T}_k$ represents a balanced tree of k levels with T_k nodes, that is, a minimal balanced tree of k levels. Then disregarding the reflections which cause $\overline{T}_k$ to fail to be unique, we see that (1) graphically constructs a minimal balanced tree of k+1 levels from given minimal balanced trees of k and k-1 levels plus one additional node and thus the relation in integers, $T_{k+1} = T_k + T_{k-1} + 1$, results by counting.

Thus we have:

Number of Levels, k	Number of Nodes in Minimal Balanced Tree, T_k
0	0
1	1
2	2
3	4
4	7
5	12
6	20
7	33
8	54
9	88
10	143
.	.
.	.
.	.

Now it is easy to show by induction that $T_k = F_{k+2} - 1$ where F_i is the i^{th} Fibonacci number, that is, $F_0 = 0$, $F_1 = 1$, and $F_i = F_{i-1} + F_{i-2}$ for $i \geq 2$.

Another proof of this fact may be given as follows.

Let G(z) be the generating function for the T_k values. That is,

$$G(z) = T_0 + T_1 z + T_2 z^2 + T_3 z^3 + \ldots \quad .$$

Then, since $T_{k+1} = T_k + T_{k-1} + 1$ for $k \geq 1$ and also,

$\frac{1}{1-z} = 1 + z + z^2 + \ldots$, we have

$$G(z) - zG(z) - z^2 G(z) - \frac{z^2}{1-z} = z \; . \qquad (2)$$

Then, by partial fraction decomposition we have

$$G(z) = \frac{z^{-1}}{1-z-z^2} + \frac{-z^{-1}}{1-z} \quad .$$

But now we know (see [7], p. 82) that $\frac{z}{1-z-z^2}$ is the generating function for the Fibonacci numbers; that is,

$$\frac{z}{1-z-z^2} = F_0 + F_1 z + F_2 z^2 + F_3 z^3 + \ldots \quad .$$

Thus, since $F_0 = 0$, $F_1 = 1$, we use the expansion of $\frac{-z^{-1}}{1-z}$ to obtain

$$G(z) = (F_2 - 1) + (F_3 - 1)z + \ldots \quad .$$

But then, equating coefficients, we obtain $T_k = F_{k+2} - 1$ for $k \geq 0$.

Now we may proceed.

Given n, a number of nodes organized in a balanced tree, Q, then if k is such that

$$F_{k+2} - 1 \leq n < F_{k+3} - 1 \qquad (3)$$

then Q has at most k levels and hence, at most k compares are needed to search Q.

Now let us solve (3) for k in terms of n.

Let m = k + 2 for brevity.

Then we have

$$F_m \leq n + 1 < F_{m+1} \; . \qquad (4)$$

But now we know (see [7], p. 82) that

$$F_m = \frac{1}{\sqrt{5}} (\phi^m - \hat{\phi}^m) \text{ for } m \geq 0, \qquad (5)$$

where

$$\phi = \frac{1}{2} + \frac{\sqrt{5}}{2} = 1.61803 \ldots$$

and

$$\hat{\phi} = \frac{1}{2} - \frac{\sqrt{5}}{2} = -.61803 \ldots \quad .$$

Then from (4) and (5) with some manipulation we have

$$\phi^m \leq (n + 1)\sqrt{5} - \hat{\phi} < \phi^{m+1} \quad . \qquad (6)$$

Hence,

$$m \leq \ln_\phi ((n + 1)\sqrt{5} - \hat{\phi}) < m + 1$$

or,

$$m = \left\lfloor \ln_\phi ((n + 1)\sqrt{5} - \hat{\phi}) \right\rfloor \quad .$$

Thus, since $m = k + 2$,

$$k = \left\lfloor \ln_\phi ((n + 1)\sqrt{5} - \hat{\phi}) \right\rfloor - 2 \quad .$$

Now as stated above, the number of compares, C, required to search any balanced tree of n or fewer nodes is such that

$$C \leq \left\lfloor \ln_\phi ((n + 1)\sqrt{5} - \hat{\phi}) \right\rfloor - 2 \quad .$$

We may approximate our derived upper bound for C as

$$\left\lfloor 2.08 \ln(2.2n + 3) \right\rfloor - 2 \quad .$$

We may also note that the shortest search path in a balanced tree of k levels has at least $\left\lfloor \frac{k + 1}{2} \right\rfloor$ nodes. This can be taken as a lower bound on the number of compares required to terminate an unsuccessful search.

7. Concluding Remarks

Several interesting problems concerning balanced tree storage and retrieval remain to be solved. One problem is the computation of the exact mean and variance of C, the number of 3-way comparisons required to search for an item. Another problem is to compute the expected number of balance operations required per insertion and per deletion taken over the construction and equilibrium existence of a balanced tree which stores items chosen from a random set of items.

The balanced tree storage and retrieval scheme has several useful features. One feature is that it is easy to retrieve the least or greatest item currently stored without needing to know the appropriate key-value. It is also easy to pass over the stored items in order by their key-values if desired.

Moreover, although hash table storage and retrieval algorithms are superior to the balanced tree scheme for symbol tables and the like, it is difficult to develop a general storage and retrieval system based on hashing methods. It is easier to program the balanced tree algorithms to deal with user-specified item formats and key-values, thus a general filing system may be developed based on the balanced tree storage and retrieval algorithms.

The nearest competitor to the balanced tree scheme as a general filing system is a sequentially organized filing scheme with auxillary tables of indices, of which the ISAM (indexed sequential access method) facilities in the S/360 operating system [5] is an example. Given a reasonable amount of insertion-deletion activity, the number of items accessed during a search may be reasonably similar in the two methods, and the balanced tree scheme uses space more efficiently. However, the fact that ISAM accesses items mostly in the same cylinder while the balanced tree scheme may not is a serious deficiency. It is possible of course that a way of building the balanced tree can be found which takes into account the difficulties involved with using movable head discs. In any event for truly random access storage mediums the problems associated with varying access time disappear and the balanced tree scheme is useful in such circumstances.

REFERENCES

(1) ADEL'SON-VEL'SKIJ, G.M.; LANDIS, Y.M. An algorithm for the organization of information. Doklady Akademii Nauk USSR, vol. 16, no. 2, p. 263:266, Moscow, USSR; also available in translation as U.S. Dept. of Commerce OTS, JPRS 17,137; and as NASA Document N63-11777.

(2) BOOTH, A.D.; COLIN, H.J.T. On the efficiency of a new method of dictionary construction. Information and Control, vol. 3, p. 334:341, December 1960.

(3) FERGUSON, DAVID E. Balanced tree searching. internal document, Programmatics Inc., January 1968.

(4) FOSTER, C.C. Information storage and retrieval using AVL trees. Proceedings of the 20th National ACM Conference at Cleveland, 1965, p. 192:205.

(5) GHOSH, S.P.; SENKO, M.E. File organization: on the selection of random access index points for sequential files. JACM, vol. 16, no. 4, p. 569:579, October 1969.

(6) HIBBARD, THOMAS. Some combinatorial properties of certain trees. JACM, vol. 9, no. 1, p. 13:28, January 1962.

(7) KNUTH, DONALD E. The Art of Computer Programming, vol. 1: Fundamental Algorithms. Addison-Wesley, Reading, Mass., 1968.

(8) KNUTH, DONALD E. Optimum binary search trees. Stanford Computer Science Dept. Technical Report CS 149, Stanford University, January 1970.

(9) WINDLEY, P.R. Trees, forests, and rearranging. Computer Journal, vol. 3, no. 2, p. 84:88, July 1960.

EFFICIENT UTILIZATION OF LIMITED ACCESS ARCHIVAL STORAGE IN A TIME SHARED ENVIRONMENT

A. M. Katcher
IBM Thomas J. Watson Research Center, Yorktown Heights, New York

ABSTRACT

The public storage in any time sharing system tends to continually grow. This necessitates the implementation of certain measures to maintain public storage. One of these possibilities is creation of an archival level of storage called "migrated" storage. Data that have not been referenced recently are moved or "migrated" to a less accessible level of external storage. Since these data are not accessed by the users directly, i.e., the data must be restored to public storage before being used, a certain variable length coding technique, viz., Huffman Coding, is used to compact and store these data. The ideas presented have been implemented on a version of TSS/360 Time Sharing System and are presently being used in a real environment. The overall compaction rate achieved was 3.16 to 1. Further details on compaction rates and timings are also presented.

KEY WORDS AND PHRASES

data compression, archival storage, Huffman Codes, data migration

I. Introduction

This paper is concerned with a specific implementation of a storage hierarchy and a means of accomplishing efficient management of this hierarchy. A certain technique will be developed along with its implementation and a statistical analysis will be given to reinforce the validity of the solution.

One thing becomes clear when discussing the permanent storage capabilities of any time-sharing system -- the resources of the system are limited by physical capabilities. It is therefore necessary to divide external storage into different classes with the charges for each class being proportional to the accessibility of data stored on that particular class. In particular, this can be illustrated using TSS/360 as an example.

On the TSS/360 System, the most important class of storage is Public Storage. Public Storage resides on direct access devices and serves as a pool which can be shared by all users of the system. However, in order to implement this pool, all the devices that comprise public storage must be on-line at system startup and must remain on-line while the system is on the air. As such, the amount of public storage is limited by the number of direct-access storage devices available, taking into account that some drives must be left over for other functions. Therefore, it seems reasonable to classify public storage on TSS/360 as a critical resource.

There is a TSS/360 System being run at the IBM T. J. Watson Research Center, Yorktown Heights, New York. It is on this system that the techniques to be described were implemented. There are approximately 350 users joined to this system and public storage consists of nine IBM 2316 disk packs. Using this as a basis and given that the system administrators do not arbitrarily erase or otherwise limit a user's amount of public storage, it was found that public storage would be growing at a rate of approximately two 2316 disk packs every three weeks. This is a situation that cannot exist for very long and, as a result, a method of controlling this growth was developed which allowed the user to retain his data on a new class of storage known as Migrated Storage (1). It is this level of the storage hierarchy which will be the concern of this paper.

Migrated Storage resides on the same medium as public storage and appears to the user to be a low-speed access extension of it. However, the migrated packs are not mounted at system startup -- only as needed. This means that there is some delay in moving a migrated data set back to public storage. In addition, users can only erase migrated data sets or restore them to public storage. They are not normally accessible while they are in a migrated status. The criterion for migrating data sets involves the desire to make as efficient use of public storage as possible. This means that data sets residing on public storage should be data sets that are currently being used. Therefore, data sets are migrated if they have not been referenced for some predetermined period of time.

The management of this storage hierarchy is divided into two basic classes -- user functions and administrative functions. The user has available three commands to manipulate his migrated data sets:

LMDS - Listed Migrated Data Sets
RMDS - Restore Migrated Data Set
EMDS - Erase Migrated Data Set

The last of these does not actually cause the data set to be erased, but flags the data set as being obsolete by specifying its erasure that night when all such data sets will be erased at one time. This obviates the necessity of constantly mounting and demounting disk packs for erasures during the user session.

The functions of the administrator are:

a. Decide when to perform a migration. This is usually done when the number of available pages in public storage falls below some "dangerous" minimum.

b. Determine for what period of time data sets may be unreferenced and still remain on public storage. This period of time will generally depend on how many pages would be freed for a given retention period.

c. Determine charges for use of migrated storage.

d. Gather statistics on the movement of data between migrated and public storage and erasures from public storage. This information will aid in developing more efficient criteria for selecting data sets to be migrated.

With this background in mind, we now come to the main concern of this paper: How can migrated storage be kept in as compact a form as possible? This greatest compaction can be achieved through the techniques of optimizing codes on the noiseless channel (2, 3, 4). One particular code construction method, discovered by Huffman (5), will be used in this implementation along with a table look-up decoding scheme.

With the large virtual storage available in TSS/360, it is now possible to efficiently implement these techniques. In addition, since migrated storage can only be reached in a limited fashion (no reading or writing of these data sets is allowed to the user) a technique for compacting this data can be developed which will achieve anywhere from a two-to-one to five-to-one compaction rate.

II. Data Compression

In general, data is represented by a fixed-length encoding, e.g., 8 bits for an EBCDIC character. However, in most data bases, not all characters occur with equal probability. It would be desirable from an efficiency standpoint to represent the most frequently occurring characters with short bit strings and the least frequently occurring characters with long bit strings. This is the basic idea behind coding on a noiseless channel.

This type of coding makes use of the statistics of the data that is being encoded. The present implementation regards each EBCDIC character as an input word and uses the frequency of occurrence of these characters to develop the code. In addition, two extra codes are included -- one denoting zero suppression and one denoting blank suppression. Since these two EBCDIC characters appear frequently in long strings, it seems reasonable to code a string of zeroes or blanks as a special character followed by an indication of the length of the string. The Huffman encoding technique is then applied to the suppressed data.

It is worthwhile to note that it is possible to use dependent statistics in developing these codes, i.e., statistics where the distribution of characters depends on the character that occurred immediately before. While this method can achieve greater compaction rates, it requires prohibitively long encoding and decoding tables and probably would need further refinements.

As an illustration of the construction of Huffman Codes, consider the example in Figure 1 consisting of an alphabet of four characters, A, B, C and D, where the characters are arranged in order of descending probability. The basic strategy is to combine the two least probable characters and add their probabilities. Then, this result is placed in proper order in the tree structure. This process continues until only one branch remains. The code for each character is determined by working back from the base of the tree to each of the probabilities. If, in this process, motion is upward from a node a "0" is assigned to the code word, otherwise a "1" is assigned. Thus, to get to the character "C", one must move down, down and up giving a code of "110" for "C".

The average code word length is computed from:

$$L = p_A \ell_A + p_B \ell_B + p_C \ell_C + p_D \ell_D$$

where p and ℓ are the probabilities and lengths of each of the characters. Making the necessary substitutions, it is seen that $L = 1.75$ bits/character whereas 2 bits/character are achieved using fixed-length coding techniques. Therefore, data compression is achieved. While this rate might not be dramatic, the fact is that greater rates are achieved when the input alphabet is larger. It is 258 characters in our case, namely, the 256 EBCDIC characters with two extra characters for zero and blank suppression.

Encoding

The procedure for encoding data is very simple once the code has been developed - it is a table lookup procedure. Since the input is fixed-length, one need only index into a table having the code word and code

CHARACTER	FIXED LENGTH CODE	HUFFMAN CODE	PROBABILITY
A	00	0	.5
B	01	10	.25
C	10	110	.125
D	11	111	.125

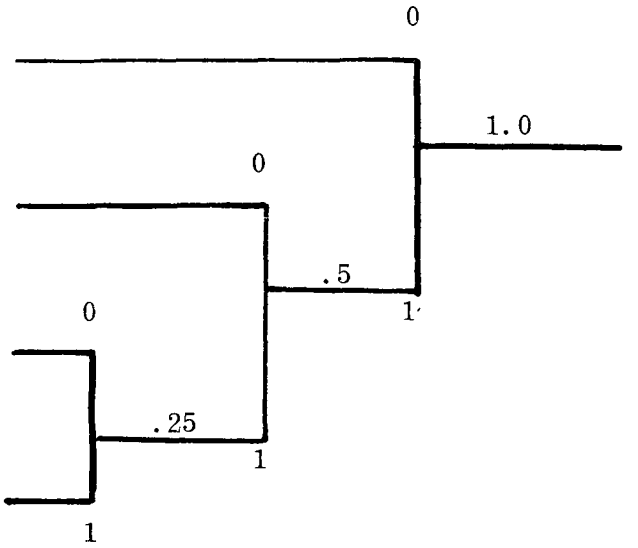


Figure 1
Example of Construction of a Huffman Code of Four Input Characters

length in an entry and insert the code word into the output record. Strings of zeroes and blanks complicate the procedure slightly but can be handled rather easily.

Decoding

Huffman Codes are uniquely decipherable, instantaneous codes. This means that a string of bits constituting a message can be decoded in one, and only one, way and that the characters can be decoded as they are received, i.e., it is not necessary to wait for the entire message to be transmitted before starting to decode it.

As the input stream is scanned one bit at a time, it is determined whether a code word has been completed. Since no code word is a prefix of any other code word, i.e., an instantaneous code, there can be no erroneous decoding of a partial code word. When a full code word is discovered in the input stream, it is decoded using a simple table look-up procedure.

When the code word decodes as a suppression character, the next eight bits are retrieved in order to determine the length of the suppressed string. This is then expanded into the output stream using a single move instruction.

At this point it might be worth mentioning another compression technique presently in use by Project INTREX at MIT (7). The technique used there is called "digramming" (8) and basically involves combining two text characters and encoding these as a one-character code. This method is limited in that it implies that not all byte configurations are legal input characters. This assumption is reasonable when the input is upper case text and numerics. In an IBM System/360 environment, this means that only 48 EBCDIC characters are valid (26 letters, 10 numerals, blank, and 11 special punctuation characters) on input, leaving 208 codes for digrams. In this limited environment, the digramming technique yields a compaction rate which is somewhat less than 2 to 1 whereas Huffman coding techniques yield significantly greater rates as will be shown later.

The TSS/360 data base consists of upper-case text, upper and lower-case text, numerical data and object code, which means that all 256 EBCDIC combinations are legal in the input stream. In addition, the digramming technique makes no provision for using suppression of strings of zeroes and blanks which account for some part of the compaction rates that were achieved in the present implementation. Timings and performance will be discussed in the next section.

III. Implementation of Compaction

Migrated data sets on TSS/360 at IBM Research, Yorktown Heights, New York (1) are being stored in compacted form. These data sets have a restricted access associated with them, i.e., users may only erase or restore them -- they cannot be read from or written into when in a migrated status. This means that the access methods of TSS are not used directly on a migrated data set.

The above described restriction implies that compaction can be implemented on these data without modifying the access methods. As such, page-size (4096 bytes) blocks of the original data set are compacted into records of the output data set which has sequential organization with a variable record length. The choice of 4096-byte blocks was prompted by the specific organization of the Virtual Access Method (VAM) of TSS/360. This access method specifies that the physical block size is 4096 bytes no matter what the logical record length, i.e., logical records span physical blocks (6).

The format and organization of the compacted data set is always sequential with variable-length records independent of the format and organization of the original data set.

This creates a problem when trying to reconstruct the original data set since it may have had an organization which was different from the compacted data set and record formats and lengths which also were different. Therefore, the first record of the compacted data set contains all relevant information pertaining to the structure of the original data set in addition to special codes so that the decompact routine will recognize a compacted data set and not attempt to decompact a data set that was never compacted in the first place.

The use of pages for the original length allows the fastest access possible to the data set because it is not necessary to go through the access methods to bring these pages into core to compact them. This procedure is done solely through the supervisor paging mechanism.

The program for performing this operation is a privileged one and can be invoked via a direct call from another module or through the commands PACK and UNPACK. These commands allow the user to voluntarily store his programs on public storage in compacted form. This will allow him to store more data in a given amount of space. Various errors are detected by the program, such as attempting to use the UNPACK command on a data set that had never been compacted. The user is informed of this fact via a message at his terminal and should take any necessary corrective action.

Statistics were gathered on the data base to be compacted and the results of these runs are shown in summary form in Figure 2. In the cases where "no suppression" is mentioned, there was no attempt to perform zero or blank suppression before simulating the Huffman Code. Where "suppression" is mentioned,

DATA SET TYPE	SUPPRESSION RATE	CODE LENGTH			COMPACTION RATE
		MIN	MAX	AVERAGE	
Partitioned (No Supp)	---	1	14	2.62	3.05:1
Partitioned (Supp)	3.22:1	3	14	6.09	4.23:1
Sequential (No Supp)	---	1	17	3.27	2.44:1
Sequential (Supp)	2.12:1	2	16	4.92	3.45:1
TOTAL (Supp)	2.39:1	3	14	6.06	3.16:1

Figure 2
Compaction for Various Types of Data Sets

TYPE	ORIG PAGES	NEW PAGES	DATA SETS	COMP RATE	CPU TIME		REAL TIME	
					ENC/ VV	DEC/ VV	ENC/ VV	DEC/ VV
VI	1986	843	76	2.36	10.0	7.8	1.13	1.04
VP	5304	1245	25	4.26	13.6	8.2	1.43	1.37
VS	293	157	12	1.87	11.0	10.7	1.19	1.28
TOTAL	7533	2245	113	3.37	12.1	8.2	1.28	1.23

Figure 3
Actual Page Compaction Rates

all strings of zeroes and blanks which were longer than 2 were suppressed and statistics taken on the data after suppression.

As can be seen from Figure 2, some statistics were taken with an attempt to classify the data set being compacted (sequential, partitioned or indexed). Data sets were not classified by content, i.e., upper case characters, upper and lower case characters, object code, etc., since TSS/360, being a general purpose time-sharing system, does not furnish this information. The only way this could have been done would be by manually examining the contents of each and every data set on the data base.

It was decided to initially compact the data base using one overall code for all types of data. At a later time, the organization or content (if it could be made available) of a data set might be used to further enhance the compaction rate of a given data set. In addition, the results with zero and blank suppression were so much better than with no suppression that the implementation was performed using zero and blank suppression. From Figure 2 it is observed that for all data (last row), zero and blank suppression gave a compaction rate of 2.39 to 1 with the final compaction rate becoming 3.16 to 1 after applying Huffman encoding. Figure 2 also gives the maximum and minimum lengths of the code words. This is a fact which affects implementation and the size of the encoding and decoding tables. For all data and suppression, the maximum code word length is 14 which means that a code word can be contained in a two byte field. If it had been 17, another byte would have been required which would have increased the size of the encoding table.

The compaction rates cited above do not really reflect the actual amount of compaction that will be achieved when the coding techniques are applied to TSS/360 data sets. This is because of the restrictions on these data sets. Any implementation requires some overhead to perform the conversion, e.g., the first record of a compacted data set contains the information about its original organization. In addition, data sets must occupy an integral number of pages (4096 bytes) so that a data set that was originally one page long will undergo no compaction since it still must occupy one page. Therefore, the amount of compaction for this scheme must be measured on a page basis.

IV. Results of Compaction

The compaction scheme was tested for public data sets on TSS/360. The statistics gathered include actual page compaction rates along with timings for the actual encoding and decoding and are presented in Figure 3. All timing comparisons were made relative to the time for performing a direct fast copying of the data set to another data set name. This copying command is "VV".

It can be seen from Figure 3 that different types of data sets compact with different compaction rates and that partitioned data sets have the greatest compaction rate. This is true because most of this compaction is achieved through zero and blank suppression. It will also be noted that Virtual Sequential data sets compact at a significantly lower rate than the other data sets. This is due to the fact that most of these data sets consist of text files which contain both upper and lower case characters. The code used is probably not efficient for these types of data sets. If a new code were to be used for these text data sets, the compaction rates should increase substantially. This, of course, would require a priori knowledge of the general type of a data set to be compacted which could be implemented in many different ways, e.g., using the data set organization for determining the appropriate code to use, or having the data set creator specify the data type as part of the creation process.

From Figure 3 it is seen that a 3:1 compaction can be achieved given that the data sets are large enough, i.e., greater than 3 pages in length. Unfortunately, this is not controllable and, as it turns out, there are quite a large number of data sets with less than 4 pages. Migrated storage was scanned and the distribution of lengths of these data sets were recorded. These results appear in Figure 4.

Assuming that all one-, two-, and three-page data sets compact into one page and that all other data sets achieve an average of 3:1 compaction, a net compaction for migrated storage becomes 2.74 to 1. This means that our migrated data base before compaction, which was 19 disk packs, became 7 disk packs.

It can be seen that although the CPU times for compaction and decompaction are significantly greater than for direct copying, the elapsed real times for these functions are only slightly greater. This is due to the large amount of in-core manipulating that must be done when performing coding. In a time sharing system, the user is more concerned with how long he must wait for a function to be completed rather than how much computing is done, and it is seen that coding does not affect the response time for the user significantly in comparison with the amount of space saved.

V. Conclusions

The method of implementing compaction described previously appears to be efficient since it compacts relatively large blocks (4096 bytes) and employs the natural physical divisions of data sets in TSS/360. In addition, the limited access imposed on the user for migrated data sets make this scheme usable even though organization of the data set is not preserved.

PAGE SIZE	NUMBER OF DATA SETS	PAGE SIZE	NUMBER OF DATA SETS	PAGE SIZE	NUMBER OF DATA SETS
1	1825	2	651	3	505
4	286	5	176	6	116
7	96	8	83	9	77
10	38	11	60	12	55
13	33	14	31	15	21
16	41	17	36	18	29
19	37	20	23	21	27
22	12	23	16	24	30
25	22	26	18	27	12
28	16	29	10	30	8
31	23	32	14	33	16
34	9	35	14	36	13
37	14	38	13	39	15
40	7	41	12	42	7
43	10	44	7	45	6
46	10	47	8	48	12
49	2	50	9	51	7
52	10	53	1	54	6
55	5	56	14	57	12
58	11	59	3	60	7
> 60	201				

Figure 4
Distribution of Data Sets on Migrated Storage

Not only does this scheme conserve migrated storage space, but it makes the job of mounting migrated disk packs much easier. Since there are more data sets per pack, the probability of having the proper pack already mounted increases and the time in which restoration of a migrated data set occurs decreases.

REFERENCES

(1) Considine, J. P. and A. H. Weis, "Establishment and Maintenance of a Storage Hierarchy for an On-Line Data Base under TSS/360", Proceedings of the FJCC, November 1969, Vol. 35, pp. 433-40, Thompson Book Co., Washington, D.C., 1969.

(2) Ash, R.T., Information Theory, pp. 27-43, Interscience Publishers, New York, 1965.

(3) Feinstein, A., Foundations of Information Theory, McGraw-Hill Book Co., New York, 1958.

(4) Fano, R.M., Transmission of Information, MIT Press, Cambridge, Massachusetts, 1961.

(5) Huffman, D.A., "A Method for the Construction of Minimum Redundancy Codes", Proceedings of the IRE, Vol. 40, No. 10, October 1952, pp. 1098-1101.

(6) Data Management Facilities of TSS/360, IBM Systems Reference Library, Form No. GC-28-2056-0, March 1970, pp. 22-25, 75-76.

(7) Project INTREX Semiannual Activity Report, PR-9, March 15, 1970, MIT, p. 28.

(8) Lesk, M.E., "Design of a Revised On-Line Information Retrieval System", Report ISR-14, Information Storage and Retrieval, Cornell University, October 1968, pp. 56-58.

DATA COMPRESSION TECHNIQUES FOR ECONOMIC PROCESSING OF LARGE COMMERCIAL FILES

James E. Mulford and Richard K. Ridall
Ridall Associates, Inc., Paoli, Pennsylvania

ABSTRACT

The application of compact coding, differencing and other techniques to indexed sequential files is discussed. The effects on system performance are discussed and reductions of almost 80% in mass storage requirements for a particular file are reported.

KEY WORDS AND PHRASES

data compression, compact coding, file compression, file storage, numeric coding, alphabetic coding, differencing, reduced alphanumeric image coding, statistical code development

I. Introduction

This paper reports on the practical application of data compression techniques to large commercial files in an IBM System/360 and DOS environment. The result of compressing one file in particular is used to illustrate quantitatively the achievable compression and its impact on the associated system. Perhaps the most significant result of this effort was not the dramatic compression achieved but the fact that it was achieved without materially altering the existing application system or programs to which the compression was applied.

Data compression, or compaction as it is sometimes called, refers simply to translating a representation of a set of data into a smaller one, without loss of information. The practical effect of this transformation is to save channel time in a data communications system or storage space in a file. The process offers potential for cost saving and other benefits which can be traded-off against cost. For example, compression has been used to reduce bandwidth (consequently power, weight and physical size) required for transmitting weather data from satellites to earth. It has also been used to reduce the size of very large files of textual information, as reported by Snyderman and Hunt.(18) It is also applicable to many commercial/industrial files containing a variety of different types of information as illustrated by the file described below.

File compression yields benefits in addition to size and consequent storage cost reduction. Because data is transferred to and from storage media (core, disk, etc.) in compressed form, program execution time is usually reduced even after making allowance for the decompression computational load. The saving in channel and core interference times are large enough in practice to provide savings even in a multiprogramming environment since CPU time is still under-utilized in most commercial applications.

In the next section, the attributes of a particular commercial file are described as background for a discussion of the compression techniques applied to it. The techniques themselves, which required no reprogramming of the original application programs, are presented next. In the last section, the results achieved from file compression are reported and performance is compared with that of the original system.

II. File and Program Environment

The file under consideration is the Master File of an application system which provides daily a list of records which meet certain search criteria to the users. There are presently only four COBOL and assembly language programs which use the Master File for all system functions including searching and file maintenance.

In uncompressed form, a Master File record is fixed length at 280 bytes, of which 20 bytes may be used as search keys. With 800,000 base records, the file is about 250 million bytes long including overflow, etc. The file is physically stored in IBM's Indexed-Sequential format to achieve the

Preceding page blank

desired efficiency in maintenance and provide random retrieval of the relatively few records that are accessed in the daily processing.

The record data consists of multiple personal names, addresses, certain legal data, amounts of money, dates, and various control codes. Within a given record, fields may or may not be present, and within a field, the data is variable length. The application is required to run under DOS which dictates that this variable length data be partitioned into fixed length records. In the uncompressed file, each base record may have zero to three trailers.

III. Compression Techniques

The techniques used for compressing the above file for disk storage fall into two groups, Logical or Statistical. They are distinguished by the data characteristic which provides the basis for compression. However, in some cases, more than one technique was applied to a particular datum.

Logical Techniques

Certain fields within a single record may be logically related, or partly redundant. For example, in our Master File there are three dates describing events which must happen in a given sequence. First, there is a date on which a legal transaction took place. Then the date on which the transaction data were entered into the Master File is recorded. If any corrections are made to a record, the date of the last correction must also be entered.

Not only must the three dates be in ascending order, but no date can be before 1940. Allowing for system life to the year 2000, the range of date fields must be less than 22,000 days. Further, it is very likely that the date of data entry will be within two days of the transaction date, so that its range can be even more restricted if it is referenced to the transaction date. A similar situation exists with the third date so that the difference between dates can be stored rather than the dates themselves. In the first instance, the difference between the actual date and 1940 is stored. Then, only the difference between the first date and the second are stored, and so forth. By representing these difference in binary, a compression from 18 bytes to less than four bytes is achieved.

This differencing technique can be applied to other fields as well. In serial files sequenced by keys such as Social Security, account, purchase order, or part numbers, the application is obvious. Differencing can also be applied to textual data, such as names, and result in a saving.

The situation is analogous to transmission of a digitized photographic (TV or whatever) image. Rather than transmitting (or storing) the absolute value of each point scanned, its difference from the last can be used with an occasional reference point included. This often results in fewer bits of information transmitted because dark and light areas are in "significant" patches rather than random.

In other cases, more special techniques were used which depended on particular relationships between fields. Generally, these special techniques were applied to fields where it was possible to derive one field from one or more others, and for infrequently appearing fields.

Statistical Techniques

Alphanumeric Coding -- Within this class of techniques one in particular was the most important contributor to data compression because it applied to the greatest number of long fields. It is called Reduced Alphanumeric Image (RAI) or compact coding technique. In it, the actual frequency of occurrence for each character (obtained by sampling the file) is used to derive a variable length binary code. To illustrate the process consider the following example:

> The RAI technique is demonstrated by a simple case where the source alphabet consists of only the four letters "A" through "D" and a space character as a separator. The probability that the next character of a string will be a particular one is assumed to be the values shown in Table I.

TABLE I.

Character, C_i	Probability, P_i
Space	0.4
A	0.2
B	0.2
C	0.1
D	0.1

A string of such characters contains a certain amount of information (E) which may be computed as follows:

$$E = \sum_{i=1}^{i=5} P_i \log_2 \frac{1}{P_i}$$

For the assumed character set, "E" is approximately 2.1 bits. That is, on the average a character conveys 2.1 bits of information. It is generally not possible to achieve this degree of compression with practical codes but it is possible to approach the limit. For example, the binary code shown in Table II will require 2.2 binary digits, on the average.

TABLE II.

Character	Code	x Probability	= Binary Digits, Average
Space	1	0.4	0.4
A	01	0.2	0.4
B	000	0.2	0.6
C	0010	0.1	0.4
D	0011	0.1	0.4
		Total Binary Digits, Average	2.2

There are many sets of codes that are just as compact. Some others, shown along with the first example, are set forth in Table III.

TABLE III.

Character	Set_1	Set_2	Set_3	Set_4
Space	1	0	00	11
A	01	10	01	10
B	000	110	10	01
C	0010	1110	110	001
D	0011	1111	111	000

Although these code sets are all equivalent in terms of compression achieved, some may be preferred in a particular situation to achieve processing efficiency. Set_2, for example, facilitates parsing a binary stream into codes using simple rules. The end of a character is denoted by a zero, or the fourth binary digit, whichever is first.

Even greater compression can be achieved if the frequency of occurrence of character pairs is also considered. To illustrate, if the letters "B" and "C" occur the same number of times as they do in the above example, but "C" always immediately follows "B", then the same character set can be coded in an average of 2.1 binary digits instead of 2.2 by including the character "BC" and eliminating the character "C".

This somewhat more advanced approach was, in fact, used for certain pairs in name fields. The format for names in both the original and compressed Master File was:

Last, first e.

Due to the format, the two pairs ",b" and ".#" occur frequently where the "b" and "#" symbols represent blanks and end-of-field respectively. It was found advantageous to code these and 10 other pairs as unique "characters". For the actual character, frequencies, probabilities and codes used, see Exhibit A1. Using this code, an average of 4.14 binary digits are required to encode one source character from a set of 43 characters. The letter frequencies found for names

differ significantly from the frequencies reported previously in english text by Nugent and Vegh (13).

Character frequencies for address data differ significantly from those of name data and a different code set was used. Some of the differences are to be expected because street addresses are usually in the format:

nnn streetname st.

where "n" is a decimal digit between 0 and 9. Note that a sequence of 2, 3, or 4 digits followed by a space or N,E,S or W would be common, so that a code prefix subset and a teletype-like shift character would provide quite compact coding. Note also that the last four characters "bst." could be expected to occur frequently and so code efficiency could be improved by coding the sequence as a single character. These expectations were verified by a sample count on address fields and the resulting code set is shown in Exhibit A2 for the prefix code subset and Exhibit A3 for the suffix subset. Taken together, these codes use 3.79 binary digits, on the average, to encode each character.

The procedure for decompressing RAI code is interesting because it is surprisingly fast and efficient. Logically, one would think of scanning the bit stream to determine the end of a character and then to translate it into the normal IBM machine code. Because of the unique design of the RAI code, however, these steps can be combined.

From Exhibit A1 another table (IV) can be constructed by using the first four bits of each code as a number and multiplying by four to form an address (see Table IV opposite).

This table is imbedded in the decompression program in a slightly different form which permits direct translation. In the case of a 4 bit code, the address in bytes allows a direct jump to the table location which contains an instruction that places the correct character to be added to the output (decompressed) character stream. In those cases where the code is more than 4 bits, the instruction placed in the table transfers control to an instruction which looks at additional bits before translating. Note that from the first four bits of the code, or table address, the code length is determined, thus avoiding iterations. Note also that since the characters are ordered by frequency of occurrence, and that characters "↑" through "M" account for almost 40% of the characters to be translated, 40% of the time the translation will be accomplished with one step.

TABLE IV.

Address	Character or Character Pair	Comment
0	↑	Four bit codes can be translated directly.
4	,b	
8	E	
12	S	
16	A	
20	b	
24	I	
28	#	
32	M	
36	L	Five bit codes require examination of the next bit in the stream.
36	C	
40	O	
40	T	
44	ER	
44	R	
48	H	
48	D	
52	.#	
52	B	
56	P	Six bit codes require examination of the next two bits in the stream.
56	G	
56	AR	
56	W	
60	IN	Etc.
60	U	
60	AN	
Etc.	Etc.	

Other Statistical Coding Techniques -- Two other techniques were used which take advantage of the statistical characteristic of the characters to be coded. Rather than using conventional 8 bit bytes or packed formats for numbers, binary codes were used where fixed length fields were desired. Greater compression resulted as compared with the RAI code technique. A sequence code, for example, was reduced from 2 bytes to 7 binary digits using this technique.

A Table-lookup technique was also used where the range of a field was much greater than the number of values it could contain. For example, certain names were in the uncompressed file which required 30 characters, yet only a few actual names existed (repeated many times). Certain types of codes also fell into this category and are recorded in the compressed file in shorthand notation. In one case, only 6 binary digits were required to store what originally required 2 bytes.

Compression Module Integration

Assembly language routines were written for each of the five techniques described above which intercept and compress data from application programs to be stored on disk. Additional routines intercept read requests and expand it into the format expected by the application program. These routines were linked to the application programs as subroutines and although this necessitated recompilation, no significant programming changes were necessary.

Compressed Record Formats

The increased use of variable length fields, partly caused by variable length binary codes, meant that selection of an optimum fixed record length was difficult. A statistical analysis was made of the cumulative effect of the various contributions to variability with the result that a base record length of 60 bytes (as compared with the original 280 bytes) was optimum and that, to a very high degree of confidence, only one trailer record would ever be required to accommodate overflow.

For simplicity, the compressed fields are placed in the new record in groups according to the compression technique employed. Since the variable and fixed length fields are segregated, there is little problem in calculating field boundaries so that the overload implied by decompressing variable length fields is small.

Record keys are similarly segregated and begin at a fixed place in the record. This is useful because most references to the file only require that the record key be known, since the most frequent task of the system is to search for matches on keys. To avoid unnecessary decompression, then, the search arguments themselves are compressed, and the search (actually performed by the ISAM modules) compares compressed data. Decompression is only performed after a "hit" has been found.

Great care was taken in the design and selection of the compression techniques applied to the record keys. Most significant was the concern for collating sequence and fixed length compression. Since ISAM requires that record keys adhere to fixed byte boundaries, there is a need to utilize techniques that will result in a fixed length compression of the record keys. The requirement to increase the compressed keys to the next byte boundary usually results in some latitude in the selection of techniques without penalty of added bytes and has made it possible to select techniques that will maintain the collating sequence of the original data. By maintaining the original file collating sequence, no special processing for conversion and file reorganization was required.

IV. Results

As mentioned above, the basic record was compressed from 280 bytes to 60 bytes, a ratio of 4.7 to 1. Or saying it another way, the record was compressed to 21% of its original volume with no loss of information. Since trailer records will be about as frequent in the compressed format as they were in the uncompressed, the total volume of the Master File has been compressed from 250 million bytes to 54 million bytes. This means that the Master File, which originally was physically stored on a Data Cell, can now be held in less than two disk drives of an IBM/2314.

This compression was obtained for two main

reasons. First, the "empty space" or blanks in the original records, primarily from unfilled fixed length fields containing variable length information, has been eliminated. Or more accurately, it has been eliminated from the individual fields and accumulated at the end of each record. This agglutination process results in less "empty" space because it is effectively shared by several fields, with longer than average fields consuming the space unused by shorter than average ones.

Second, the information stored in a record requires less space because more compact coding has been used. Overall, an average of 4.03 binary digits are required to record a character in name and address fields in the compressed record, as compared with almost 8* in the original Master File. Codes, amounts and other special fields required even fewer binary digits per character.

* Less than 8 binary digits were required because some fields used a packed format.

In addition to the elimination of empty space within fields, some fields were eliminated in their entirety. It was found that certain fields contained redundant information which existed in other fields. Often, the redundant information was generated within the system prior to storing the data rather than at output time. The data elimination in these fields is technically trivial, but produces small but significant results since reduction to zero represents infinite compression and major storage savings. Data compression studies usually act as a catalyst and result in constructive user attitudes and significant reductions in record content by elimination of unneeded information identified and volunteered by the users themselves.

A summary of the compression achieved is shown in Table V. The three general categories mentioned above (empty space, reduced representation, and elimination) are shown with the subdivision into the various techniques comprising reduced representation. The reader may note that it was not possible to eliminate all empty space since the DOS-ISAM environment requires fixed length records.

TABLE V. SUMMARY OF COMPRESSION

Technique	Original Storage In Bytes		Compressed Storage In Bytes		Net Savings In Bytes		Savings as % of Original Record
Eliminated Empty Space		123.6		6.6		117.0	41.8
Reduced Representation							
RAI Coding	67.4		34.0		33.4	11.9	
Decimal to Binary	25.0		8.5		16.5	5.9	
Table Look-up	14.0		5.9		8.1	2.9	
Differencing	24.0		5.0		19.0	6.8	
Sub Total		130.4		53.4		77.0	27.5
Field Elimination		26.0		0.0		26.0	9.3
Total Compression		280.0		60.0		220.0	78.6

Operationally, programs which originally required about one hour to run, require only about 10 minutes with the compressed file. Of course, part of this improvement comes from use of a faster storage device made economically possible by compression. If the device component of the performance increase is removed, it is estimated that the performance of the system would be about the same as with the original file. That is, the compression and decompression CPU overhead is offset by improved channel and buffer performance.

The assembly language compression routines were written carefully to avoid waste space since limited core was available. As a result, the total area required for the five techniques, tables, and working storage were approximately 5k bytes.

In conclusion, this compression effort is considered successful from technical and economic viewpoints and demonstrates the feasibility of retroactively applying data compression techniques in a commercial environment.

REFERENCES

(1) DE MAINE, P.A.D.: SPRINGER, G.K. The COPAK Compressor. In: File Organisation. Selected papers from File 68 -- An IAG Conference, Swets and Zeitlinger N.V., Amsterdam, 1969, 149-158.

(2) GAINES, HELEN FOUCHÉ. Cryptanalysis, Dover, New York, 1956.

(3) KAHN, DAVID. The Codebreakers, MacMillan Company, New York, 1967.

(4) KALLAB, J. A linear geographical code for management information systems. In: Computers and Automation. 17, 4 (April 68) 24-30.

(5) KENYON, W.S. Errors in Transmission of Compressed Data, PhD Thesis, Princeton Univ., Princeton, New Jersey, 1968.

(6) KERPELMAN, C. (Ed.) Proposed American National Standard; Identification of States .. ...for Information Interchange. In: Communications of the ACM, Vol. 13, No.8 (Aug.70), 514-515.

(7) KUKENHEIM, L. Coding of grammatical data. In: Sprachkunde und Informationsverarbeitung. No.1, 1963, 45-47 (German).

(8) LOHSE, E. (Ed.). Data code for calendar date for machine-to-machine data interchange. In: Communications ACM. 11, 4 (April 68) 273-274.

(9) MARRON, B.A.: DE MAINE, P.A.D. Automatic data compression. In: Communications of the A.C.M. 10, (November 1967), 711-715.

(10) MAURER, W.D. File compression using Huffman coding. In: Computing Methods in Optimization Problems, Vol.2, 247-256. Second International Conference on Computing Methods in Optimization Problems, San Ramo, Italy, September 1968, Academic Press, New York 1969.

(11) NEUMANN, P.G. Efficient Error-limiting Variable-length Codes. Thesis, Harvard University, Cambridge, Mass., 1961.

(12) NUGENT, W.R. Compression word coding techniques for information retrieval. In: Journal Library Automation 1, 4 (Dec.68), 250-260.

(13) NUGENT, W.R.: VEGH, A. Automatic word coding techniques for computer language processing. In: RADC-TDR-62-13, Vols. 1, 2.

(14) OVERHAGE, CARL F.J. (Ed.) Project INTREX Semiannual Activity Report PR-9, March 15, 1970, M.I.T., Page 28.

(15) PRATT, FLETCHER. Secret and Urgent, The Story of Codes and Ciphers, Blue Ribbon Books, Garden City, New York, 1939.

(16) SCHWARTZ, E.S. A Dictionary for Minimal Redundancy Encoding, JACM 10, (1963) 413-439.

(17) SCHWARTZ, E.S.: KLEIBOEMER, A.J. A language element for compression coding, In: Information and Control, 10, 3 (March 67) 315-333.

(18) SNYDERMAN, MARTIN: HUNT, BERNARD. The myriad virtues of text compaction. In: Datamation, 16, 16(1 December 70), p. 36.

(19) VERHOEFF, J. Error detecting and correcting codes for the decimal number system. In: Proceedings International Symposium on Automation of Population Register Systems, Vol. 1, 447-454.

(20) WRIGHT, M.A. Mechanizing a large index; appendix: the soundex code. In: The Computer Journal, 3(July 1960), p. 83.

EXHIBIT A1

SOURCE ALPHABET AND CODES
USED FOR NAME FIELDS

Character	Frequency	Probability	Code	Character	Frequency	Probability	Code
↗		.08726	0000	J	4,669	.01344	1111110
,b	21,654	.06231	0001	ON	4,204	.01210	1111111
E	21,572	.06208	0010	IL	4,005	.01152	0000
S	17,718	.05099	0011	K	3,927	.01130	0001
A	17,381	.05002	0100	F	3,412	.00982	0010
b	16,914	.04867	0101	EL	3,372	.00971	0011
I	14,523	.04180	0110	OR	3,227	.00928	0100
#	14,309	.04117	0111	EN	3,084	.00888	0101
M	13,990	.04026	1000	V	2,666	.00767	0110
L	13,779	.03966	10010	-	2,381	.00685	0111
C	12,078	.03475	10011	Z	2,002	.00576	1000
O	11,856	.03412	10100	Ø	1,000	.00288	10010
T	11,853	.03411	10101	X	499	.00143	10011
ER	10,967	.03156	10110	7	123	.00035	10100
R	10,860	.03125	10111	Q	117	.00033	10101
H	10,747	.03093	11000	Ø	102	.00029	10110
D	10,318	.02969	11001	1	97	.00028	10111
.#	9,701	.02792	11010	9	51	.00015	11000
B	7,532	.02167	11011	6	24	.00007	11001
P	7,319	.02107	111000	'	221	.00064	11010
G	7,191	.02069	111001	3	19	.00005	11011
AR	7,014	.02018	111010	5	16	---	111000
W	6,122	.01762	111011	4	15		111001
IN	5,983	.01721	1111000	2	15		111010
U	5,768	.01660	1111001	8	6		111011
AN	5,626	.01619	1111010	&	1		1111000
Y	5,414	.01558	1111011	.	-		1111001
N	5,203	.01498	1111100				
AL	4,850	.01395	1111101	Totals	347,497	.99969	

EXHIBIT A2

PREFIX CODES
USED IN ADDRESS FIELDS

Character	Frequency	Probability	Code
b	29,500	.1965	000
↗	24,010	.1600	001
1	14,693	.0980	010
2	12,945	.0863	0110
3	9,196	.0612	0111
.	9,000	.0600	1000
5	7,907	.0527	1001
0	7,639	.0508	1010
4	7,319	.0487	1011
6	5,858	.0397	11000
7	4,607	.0307	11001
8	4,229	.0282	11010
N	4,000	.0267	11011
9	3,534	.0235	11100
W	1,500	.0100	11101
S	1,500	.0100	11110
E	1,500	.0100	11111
Totals	148,937	.9930	

EXHIBIT A3

SUFFIX CODES -- USED FOR ADDRESS FIELDS

Character	Frequency	Probability	Code
E	18,824	.0858	0000
R	15,931	.0726	0001
T	15,866	.0723	0010
N	14,489	.0660	0011
A	13,633	.0622	0100
bST.#	12,500	.0570	0101
b	12,220	.0557	0110
O	11,898	.0542	0111
S	11,196	.0510	1000
L	9,875	.0449	10010
#	8,510	.0387	10011
D	8,416	.0383	10100
↗		.0360	10101
I	7,748	.0352	10110
H	7,397	.0337	10111
C	5,021	.0228	11000
M	4,256	.0194	11001
W	3,727	.0170	11010
G	3,509	.0160	11011
U	3,449	.0157	111000
P	3,278	.0149	111001
B	3,149	.0143	111010
bAVE.#	3,000	.0137	111011
K	2,601	.0118	1111000
.	2,430	.0111	1111001
Y	2,354	.0107	1111010
F	2,235	.0102	1111011
V	1,702	.0078	1111100
,	974	.0044	1111101
0	800	.0036	1111110
1	800	.0036	1111111
2	800	.0037	0000
3	800	.0036	0001
4	800	.0036	0010
5	800	.0037	0011
6	800	.0036	0100
7	800	.0036	0101
8	800	.0037	0110
9	800	.0036	0111
J	441	.0020	1000
Z	410	.0019	10010
-	257	.0012	10011
X	249	.0011	10100
Q	121	.0005	10101
@	42	.0002	10110
&	00	.0000	10111
Totals	219,708	1.0006	

OPTIMAL CLASSIFICATION AND ITS CONSEQUENCES

David M. Jackson
Department of Computer Science
Cornell University, Ithaca, New York

ABSTRACT

A particular classification and retrieval model are considered. A notion is introduced which indicates the extent to which retrieval performance may be improved by a suitable choice of classification within the model. A method for determining the optimal performance for the model is outlined together with an algorithm for constructing the classification which allows this limit to be attained. A treatment of the mathematical preliminaries for a particular class of match function is given. The relevance of the analysis in research on information retrieval systems is discussed.

KEY WORDS AND PHRASES

retrieval model, classification, optimum, optimum classification, evaluation, match function.

1. Introduction

A number of experiments have been performed to examine the application of term classifications to Information Retrieval. Salton (12) and Sparck Jones and Jackson (15), among others, have shown that a certain measure of improvement in performance over simple term retrieval may be obtained by using classifications generated automatically. Both groups have observed empirically that the classifications must have certain macroscopic characteristics if improvement in retrieval performance is to be obtained. The homogeneity of classes and the smallness of classes with respect to the number of terms of the vocabulary are examples of such characteristics. Clearly, hypotheses concerning classification validated in experimental studies are of value as design criteria in the manual or automatic construction of word classifications. It is the intention of classification research, at least in experimental studies in Information Retrieval, to determine the models and methods which give in general the best performance, and to characterize the factors which account for the allocation of terms to particular classes.

Practical constraints must be taken into account. For example detailed semantic and syntactic information is too costly to extract for a large amount of material. Accordingly, it has become common practice to restrict admissible documentary data to a knowledge of the unordered sets of keywords or terms (possibly differentiated by probabilistic weights) which serve as "characterizations" of each document of the collection. In order to recover in part some of the semantic information which has been neglected, some form of semantic hypothesis (Doyle (3)) is invoked. It is assumed that terms which cooccur in a significant number of documents of the collection are related synonymously (the relation S^*) or topically (the relation T^*), and that the prime goal of classification is accordingly to produce automatically a grouping of terms into sets of synonyms or topically related terms. Such a notion, if it is supported by evidence from implemented retrieval systems, is valuable. It allows us to assert that the terms used in characterizing documents form a heterogeneous multivariate population which may be decomposed into "intelligible" classes. The classes are intelligible in the sense that a name may be assigned to the extensions of each class, and that this name enables us to determine by synonomy or by topical relatedness whether other terms of the population should have been assigned to the class. That is, the assumption provides us with a test for the "goodness" of a classification. The purpose of this paper is to examine the assumption more critically for a particular model of classification.

2. Limiting Performance of a Retrieval System

For expository purposes we shall consider, in Section 4, a simple retrieval system, but one which might be made more sophisticated without difficulty. An account of the system has been given elsewhere (Jackson (4)). For experimental purposes, we shall presuppose a means of evaluating retrieval performance and assume the availability of a base set of requests and a list of documents relevant to each request. It will also be assumed that recall and precision statistics are adequate for evaluating retrieval performance, although this has been called in question by Taube (17). A number of techniques for evaluation are available ((13), (1), (16)). These techniques, if the evaluation is to be regarded as a real rather than an accidental property of the system itself, rely

Preceding page blank

on the supposition that there is some consistency in the judgement of relevance. Resnick and Savage (11) confirm this supposition by experiment. Further support for the use of properly computed relevance judgements is afforded by Salton (14). Finally, it will be assumed that term descriptions of the documents of the collection are available and that morphological variants have been conflated to canonical form. The manner in which this is done will not concern us here.

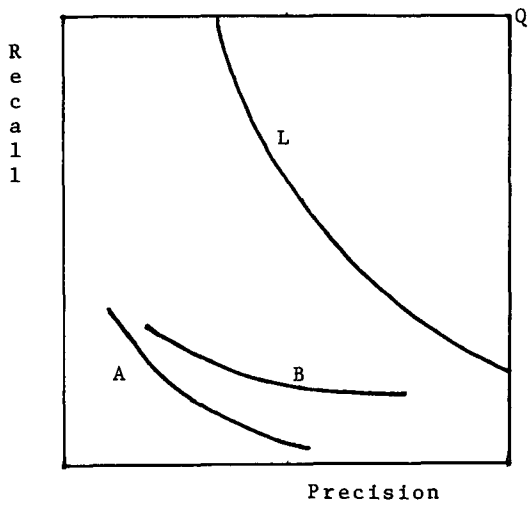
Methods for classifying terms or keywords according to their occurrence in documents are numerous. Models of classification are also numerous and include hierarchies, non-hierarchical classifications with probabilistic assignment and non-hierarchical overlapping classifications with non-probabilistic (all-or-none) assignment. Nagy (10) gives a review of a wide variety of classification techniques of particular relevance in pattern recognition, but nevertheless of value in Information Retrieval. A detailed account of the function of classification in Information Retrieval is given by Jackson (8).

Attention is confined to non-hierarchical classifications of terms. The presence of a term in a number of classes may be regarded as a means of differentiating the multiple senses of a term. The presence of a term in a class is not quantified. This model of classification may be regarded as an extension of the association lattice (2). Sets of terms with a common immediate node in the lattice map into the classes of the classification, and the order relation which induced the lattice is omitted. The classification is used to provide expansions of both documents and requests. Thus although terms from a request and document may not match lexigraphically, they may nevertheless belong to the same class, and this may be utilized as a contribution to their correlation. The task of expansion and correlation is accomplished by a "match function".

All classifications of the model may be represented as binary arrays, whose rows signify documents, whose columns signify terms and whose elements, zero or one, indicate the absence or presence of a term in a class. If upper limits are set on the dimensions of this matrix then all classifications of the model may be enumerated since the number of differing binary arrays with these dimensions is finite (but large). Indeed, each classification may be tested, in principle, in the retrieval system to determine the classification which yields the best performance. Such a classification will be called a limiting classification of the model and is denoted by C_L . A arbitrary classification of the model is called a pseudo-classification. A discussion of the predictive and evaluative uses of pseudo-classifications is given by Jackson (14).

3. Properties of the Limiting Classification

The following assertions follow from the construction of the limiting classification C_L :



Legend:
i) A and B are performance curves for classification algorithms π_A and π_B .
ii) L is the performance curve resulting from the use of the limiting classification C_L .
iii) Q is the paradigm performance.

Retrieval Performance Curves for a Retrieval System Using Different Classification Algorithms.

Figure 1.

If L is the performance curve for the system based on the limiting classification then no classification of the model will yield better performance in the sense that the performance of π_B is better than π_A . In particular, since classifications produced by standard algorithms π are a subset of the set of all pseudo-classifications of the model, then no algorithm exists which will generate a classification yielding retrieval performance better than L .

If A is the performance curve when a classification generated by an algorithm π_A is used in the retrieval system, then the disparity between A and L gives an indication of the extent to which π may

be improved by a more sophisticated treatment of the classification problem in Information Retrieval.

If Q is the paradigm performance, then L may differ from Q . The relevance judgements may be such that conflicting assignments of terms to classes are required to obtain paradigm performance.

The use of pseudo-classifications enables a number of goals to be conceptualized. Examination of C_L enables us to corroborate or deny the empirically based macro characteristics of classifications, alluded to in Section 1. An eventual aim is to confirm or deny that C_L is a classification of terms into topic classes or synonym classes, which is a consequence of the semantic hypothesis. This is a legitimate procedure since C_L is constructed without invocation of this hypothesis. A more tangible aim is to confirm or deny that the classes of C_L are characterized by strong internal homogeneity, computed on the basis of pair-wise similarities between terms. The final goal is to determine the micro characteristics of the classification, namely a replacement for the assumption of synonomy or topical relatedness between pairs of terms of a class. This question may be regarded as a problem of determining an analogical relationship, R* , between the terms such that

$$(\forall C_m \in C)(\forall t_i,t_j,t_k,t_l \in C_m)(t_i:t_j \underset{R^*}{:} t_k:t_l)$$

where the t's are terms in classes C_m of a classification C . Stated in this way, the problem is suggestive of the geometrical analogy problems treated by Evans (4), although there are no immediately apparent primitives for defining the relationship R* in the retrieval case. If S* or T* are substituted for R* then we obtain the property of C which is customarily assumed in Information Retrieval. Confirmation of the property of strong internal homogeneity or determination of the relationship R* , if it exists, will permit the construction of C_L without reference to the relevance judgements.

4. The Retrieval System and Its Adjustment

Let $\underline{D}$, $\underline{Z}$, $\underline{R}$, and $\underline{C}$ be rectangular arrays. $\underline{D}$ is the document array, containing the index descriptions, $\underline{Z}$ is the relevance array giving the relevance judgements between documents and requests. $\underline{R}$ is an array containing the terms descriptions of the requests. $\underline{C}$ is an array specifying a classification of the model. Documents are retrieved if the value of the match function for a specific document-request pair is greater than a prescribed threshold, ℓ_o . Both the term descriptions of the document and request and the expansions of these by the classification are used in computing the match between D and R . We write in general $\ell_{ij} = M(D_i,R_j,f(D_i,\underline{C}),f(R_j,\underline{C}))$ where $f(D_i,\underline{C})$ gives the set of all classes of $\underline{C}$ which have at least one term in common with D_i . D_i is a document description and R_j is a request description. Clearly, comparison of $f(D_i,\underline{C})$ and $f(R_j,\underline{C})$ permits recovery of a non-lexicographic match provided the terms belong to the same class. Denote $f(D_i,\underline{C})$ and $f(R_j,\underline{C})$ by D_i' and R_j' respectively, their dependence on $\underline{C}$ being made implicit. Let T and T' be contingency tables giving the incidence of terms and classes, respectively, of an arbitrary document D_i with an arbitrary request R_j . Then T and T' are given, with their marginal sums, by:

T:

	D_i	$\bar{D}_i$	
R_j	a	b	n
$\bar{R}_j$	c	d	N-n
	m	N-m	

T':

	D_i'	$\bar{D}_i'$	
R_j'	a'	b'	n'
$\bar{R}_j'$	c'	d'	N'-n'
	m'	N'-m'	

where N is the number of terms in the vocabulary and N' is the number of classes in the classification $\underline{C}$. Thus, we may write $\ell_{ij} = M(a,b,c,a',b',c')$. The strategy for constructing C_L is as follows. If $Z_{ij} = 1$ (i.e. D_i relevant to R_j) then $\underline{C}$ is adjusted so that $\ell_{ij} \geq \ell_o$. If $A_{ij} = 0$ (i.e. D_i not relevant to R_j) then $\underline{C}$ is adjusted so that $\ell_{ij} < \ell_o$. The adjustment of $\underline{C}$ is accomplished by a sequence of perturbation functions.

Two operations will be used for constructing C_L . These are the assign function $a(x \rightarrow y);$<conditions on x,y> which assigns a term x to a class y ; and the remove function $r(x:y \rightarrow z);$<conditions on x,y,z> which removes a term x from a class y and assigns it to a class z . The classes y and z may either be classes existing in the classification or null-classes to which no assignments have been made. While under construction, $\underline{C}$ is a three-valued array. C_{ij} may have the values 1 , 0 , -1 signifying that t_i belongs to C_j , that t_i may or may not belong to C_j , and that

t_i does not belong to C_j . The action of $a(t_i \rightarrow C_j)$ is $C_{ij} = 1$ while the action of $r(t_i:C_j \rightarrow C_k)$ is $C_{ki} = 0$, $C_{kj} = 1$.

The conditions applying to the arguments of the a and r functions ensure that their action is consistently to increase or decrease ℓ_{ij} or to leave ℓ_{ij} unchanged. Accordingly, the perturbation functions may be categorized, ab initio, by their effects on ℓ_{ij} . For a particular D_i and R_j and relevance judgement Z_{ij} , a perturbation is selected from among those which have the required effect on ℓ_{ij} . Thus C_L is constructed iteratively from a knowledge of the relevance judgements. The following section provides an analysis of the a and r functions. The analysis is applied in Section 7 to a specific match function. A single step in the construction is given in an example showing the evaluation of the conditions on the arguments of the a and r functions.

The problem of constructing limiting classifications may be regarded as one of optimizing an objective function globally over subsets of vertices of a hypercube. The objective function is a measure of argeement between the documents which are retrieved and those which have been judged relevant. The problem of optimizing a function globally over the vertices of a hypercube is encountered, for example, in hierarchical classification (Hartigan (6)), and is solved generally under the assumption that the solution is accessible in a sequence of steps, each locally optimal. This assumption will be made in connection with limiting classification, where a step in the construction corresponds to the application of a perturbation function which optimizes the objective function. Gleason (5) comments interestingly on the general validity of this assumption in problems of discrete optimization.

5. Mathematical Preliminaries

The remove operation consists of two sections which may be separated and considered independently. The actions will be called "place" and "take" and are performed by the p-function and the t-functions respectively. $t(t_k,C_i)$ has the effect $C_{ki} = 0$, while $p(t_k,C_j)$ has the effect $C_{kj} = 1$. The functions are related by $a(x \rightarrow z) \equiv p(x,z)$ and $r(x:y \rightarrow z) \equiv t(x,y) \cdot p(x,z)$.

In considering the effect of the t and p functions on the class-description B of a terms description A of a document or request, two cases are to be distinguished. Suppose that a term is taken from a class C which possesses no other terms in common with A . The class description of A will be altered by the complete removel of the class C from it and this will have an effect on a' , b' and c' . When the term is subsequently placed in another class there will be a further change in a' , b' and c' . The combined effect of these two changes is the effect produced by the remove operation. Suppose, however, that a term is taken from a class which has another term in common with the term description A . There will be no change in the values of a' , b' and c' although the subsequent placing of the term in another class may produce one. The terms of A and the classes of B must therefore be grouped according to the effect of the t-function on them. Two conditions are required; one is to test whether a specific term, if taken from the class y , will cause that class to be omitted from the class description of A ; the other is to test whether there will be no such change. These conditions are respectively $L_1(y,A) \equiv (|y \wedge A| = 1)$ and $L_2(y,A) \equiv (|y \wedge A| \neq 1)$ and have the values $\underline{T}$ (true) or $\underline{F}$ (false). They are related by $L_1 = \overline{L}_2$ but it is convenient to preserve the separate identity of the conditions. There are therefore two forms of the t-function. These are $t_1(x,y) \equiv t(x,y);L_1(y,A)$ and $t_2(x,y) \equiv t(x,y);L_2(y,A)$.

Suppose that X is a subset of A and Y is a subset of B where B is the class description of the term description A with respect to the classification. Suppose also that x is a term belonging to the class y . X and Y are defined by:

5.1 i) for any term x belonging to X , there exists a class y belonging to Y such that if x is taken from y then y will vanish from B .

ii) for any class y belonging to Y , there exists a term x belonging to X such that if x is taken from y then y vanishes from B .

iii) if the class y vanishes from B when x is taken from y , then x belongs to X and y belongs to Y .

X is called the <u>domain</u> in A of the t_1-function. Y is called the <u>range</u> in B of the t_1-function.

Suppose that the states of the classification before and after the application of $t_1(x,y)$ are $\underline{C}$ and $\underline{C}'$ respectively. It follows from the definition of X and

Y , that X and Y have the properties:

5.2 i) $x \in X \Rightarrow \exists\, y \in Y \,.\, y \notin f(A,\underline{C}')$

ii) $y \in y \Rightarrow \exists\, x \in X \,.\, y \notin f(A,\underline{C}')$

iii) $y \notin f(A,\underline{C}') \Rightarrow x \in X$, $y \in Y$
where $X \subseteq A$, $Y \subseteq B$ and
$B = f(A,\underline{C})$

iv) $X = (\forall x)(\forall y)(x \mid x \in A \wedge y; y \in B; L_1(y,A)) \equiv \underline{domain}(t_1;A;B)$

v) $Y = (\forall y)(y \mid y \in B; L_1(y,A)) \equiv \underline{range}(t_1;A;B)$.

The domain in A and the range in B of the t_2-function are defined in a similar way. The effects of the t-functions may now be stated explicitly. $t_1(x,y)$ decreases the cardinal of the range of t_1 by one iff x is in the domain in A of t_1 and y is in the range in B of t_1, and has no effect otherwise. $t_2(x,y)$ has no effect on the cardinal of the range in B of t_2 if x belongs to the domain in A of t_2 and y belongs to the range in B of t_2 .

Let $Y_i = \underline{range}(t_i;A,B)$ and $X_i = \underline{domain}(t_i;A,B)$, i = 1,2 . Then Y_1 and Y_2 are disjoint and cover B . X_1 and X_2 are not disjoint but cover A .

6. Effect of the Place and Take Functions on a Document-Request Pair

Suppose that the terms of R and D and the classes of R' and D' are partitioned into the sets defined as follows.

$F = R \wedge D$, $G = D \wedge (\overline{R \wedge D})$,

$H = R \wedge (\overline{R \wedge D})$, $F' = R' \wedge D'$,

$G' = D' \wedge (\overline{R' \wedge D'})$ and

$H' = R' \wedge (\overline{R' \wedge D'})$. Then from Section 3, $|F| = a$, $|G| = c$, $|H| = b$, $|F'| = a'$, $|G'| = c$; and $|H'| = b'$. The analysis of Section 5 will now be applied to partition F' . Let $F_1' = \underline{range}(t_1;R,F')$ and $F_2' = \underline{range}(t_2;R,F')$. Then F_1' and F_2' are disjoint and cover F' . Let $F_1'' = \underline{domain}(t_1;R,F')$ and $F_2'' = \underline{domain}(t_2;R,F')$. Analogous statements hold for an exhaustive and disjoint partition of H' into H_1' and H_2' with H' replacing F' in the above. Similarly F' in D' may be partitioned into the disjoint sets F_3' and F_4' which cover F' . These are given by $F_3' = \underline{range}(t_1;D,F')$, $F_4' = \underline{range}(t_2;D,F')$, $F_3'' = \underline{domain}(t_1;D,F')$ and $F_4'' = \underline{domain}(t_2;D,F')$. Analogous statements hold for the exhaustive disjoint partition of G' into G_1' and G_2' . The effect of the t-function on (D,R) will be examined.

Suppose that x belongs to F_1'' . Then by 5.2(i), a class y can be found such that if x is removed from y then the cardinal of F' will be reduced by one. There is, however, another effect if x also belongs to F_4'' . A class will remain in D' which matched with the class in F' which vanished with the removal of x , and this class will be unaffected by the removal. It will therefore become attached to G' , with the result that the cardinal of G' will be increased by one. The cardinal of H' will be unaffected. If x does not belong to F_4'' but to F_3'' , then the cardinal of G' will be unaffected. The remaining effects on F' , G' , H' may be determined similarly. If the increments in a' , b' and c' due to the t-function are denoted by $\Delta_t a'$, $\Delta_t b'$, and $\Delta_t c'$ respectively, then their values are given in Table 1. A table giving the effects of the p-function may be derived in the same way. Finally, the effects of the r-function and the a-function may be determined from the tables for the t-function and the p-function since $a(x \to z) = p(x,z)$ and $r(x:y \to z) = t(x,y) \cdot p(x,z)$.

X	$\Delta_t a'$	$\Delta_t b'$	$\Delta_t c'$
F_{14}''	-1	0	1
F_{23}''	-1	1	0
F_{31}''	-1	0	0
F_{42}''	0	0	0
H_1''	0	-1	0
H_2''	0	0	0
G_1''	0	0	-1
G_2''	0	0	0

where $F_{ij}'' = F_i'' \wedge F_j''$.

Effects of $t(x,y); x \in X$ on a' , b' and c' .

Table 1

7. Application to a Particular Match Function

The match function which will be used in this example is intended for illustrative purposes. It is, however, suficiently complex to provide a good application of the analysis given in Section 5. The match function is

$$\ell = (a/n) + (a' - b')/n' .$$

The contribution to ℓ from term matches is a/n while the contribution from class matches is $(a' - b')/n'$. Denote these by ℓ_t and ℓ_c respectively. Now $\ell_c = 1 - 2b'/n'$ since $a' + b' = n'$. Thus $\Delta_r \ell = \Delta_r \ell_c$ (since the term descriptions of R and D remain constant under r). Thus:

$$\Delta_r \ell = -2(n'\Delta_r b' - b'\Delta_r n')/(n'(n' + \Delta_r n') = 2.g(\Delta_r b', \Delta_r n', b', n') \text{ , say.}$$

Suppose that the effect of $t(x,y);x \in X;y \in Y$ on a', b' and c' is $\Delta_t a'$, $\Delta_t b'$ and $\Delta_t c'$. Suppose also that the effect of $p(x,z);x \in Z';z \in Z$ on a', b' and c' is $\Delta_p a'$, $\Delta_p b'$ and $\Delta_p c'$. Then the effect of $r(x:y \to z);x \in X";y \in Y;z \in Z"$ is given by $\Delta_r a' = \Delta_t a' + \Delta_p a'$ and similarly for b' and c', provided $X" \subseteq X \cap X'$ and $Z" \subseteq Z$. Similarly, the effect of $a(x \to z);x \in X';z \in Z$ is given by $\Delta_a a' = \Delta_p a'$ and similarly for b' and and c'. The values of $\Delta_r b'$ and $\Delta_r n'$ may be determined from the above for each r-function and for each a-function. It may be shown that the pairs of values of these increments which occur for this particular match function are $(1,1)$, $(1,0)$, $(0,-1)$, $(0,0)$, $(-1,-1)$, $(-1,0)$ and $(0,-1)$. The increases in ℓ corresponding to these changes are given below, together with the grouping of the associated perterbations into <u>increasing</u>, <u>decreasing</u> and <u>level</u>.

$g(1,1,b',n')$, $g(1,0,b',n')$, $g(0,-1,b',n')$ are positive and give <u>decreasing</u> perturbations.

$g(0,0,b',n')$ is zero and gives <u>level</u> perturbations.

$g(-1,-1,b',n')$, $g(-1,0,b',n')$, $g(0,1,b',n')$ are negative and give <u>increasing</u> perturbations.

The following example illustrates one step in the construction of limiting classifications. We assume for the purposes of the example that the document D is known to be relevant to the request R and that the value of ℓ_o, the threshold of the match function below which documents are not retrieved, is 0.9. The step is locally optimal in the sense that the perturbation function increases the match between R and D to a value above the threshold ℓ_o. A discussion of the complete set of perturbation functions from which this particular one has been selected is omitted for brevity. However, the fact that its effect (whether <u>increasing</u>, <u>level</u>, or <u>decreasing</u>) on the match function may be predicted in advance, thereby providing the means for selecting it from amongst the other perturbation functions, has been discussed above.

Consider a vocabulary of four terms and a classification of three classes. Suppose that $R = (1001)$ is a request and that $D = (1110)$ is a document, and that the present state of the classification is:

$$C = \begin{bmatrix} 1000 \\ 0000 \\ 0001 \end{bmatrix} \quad \text{so} \quad N = 4 \quad \text{and} \quad N' = 3 .$$

Then $F = (1000)$, $G = (0110)$, $H = (0001)$ so $a = 1$, $b = 1$ and $c = 2$. In addition $R' = (010)$, $D' (100)$ so $F' = (100)$, $G' = (000)$ and $H' = (001)$ whence $a' = 1$, $b' = 1$ and $c' = 0$. The value of the match between R and D, given the classification C, is $\ell = 0.5$.

It is known from prior categorization that the function $r(x:y \to z);x \in R \wedge y;y \in F';L_1(y,R);z \in H$ increases the match value. Each of the conditions separated from the function and argument list by semi-colons must be satisfied by the argument triad (x,y,z) before the function may be applied to the classification C. Violation of any of the conditions invalidates the function. To determine the possible value of x, y, z we proceed as follows, verifying the validity of each condition in succession:

i) <u>$y \in F'$</u>

$y \in (100) \Rightarrow y = C_1$

ii) <u>$x \in R \wedge y$</u>

$x \in (1001) \wedge (1000) = (1000) \Rightarrow x = t_1$

iii) <u>$z \in H$</u>

$z \in (001) \Rightarrow z = C_3$

iv) <u>$L_1(y,R)$</u>

$\equiv (|y \wedge R| = 1)$

$\equiv (|(1000) \wedge (1001)| = 1) \equiv$ <u>true</u> .

Thus all conditions are satisfied for the triad (t_1, C_1, C_3). The effect of this perturbation on C is to update C to C' where:

$$C' = \begin{bmatrix} 0000 \\ 0000 \\ 1001 \end{bmatrix}$$

For this classification, $R' = (001)$ and $D' = (001)$ so the match between R and D is 1.50. This confirms a predicted increase in the match value for the given R and D and demonstrates how it is achieved by modification of the term classification.

Similarly, it is known from prior categorization that the following function decreases the match between R and D: $r(x:y \to z);x \in F \wedge y;y \in F';L_2(y,R);L_1(y,D);z \in V$

where V is the set of null classes in the classification. Suppose that the state of

the classification is now C', obtained from C after the application of the previous perturbation. To determine the possible values of x, y, z we proceed as before:

i)' $\underline{y \in F'}$

$y \in (001) \Rightarrow y = C_3$

ii)' $\underline{x \in F \wedge y}$

$x \in (1000) \wedge (1001) = (1000) \Rightarrow x = t_1$

iii)' $\underline{z \in V}$

$z \in V = C^* \wedge \overline{(D' \vee R')}$

$= (111) \wedge \overline{((001) \vee (001))}$

$= (110) \Rightarrow z = C_1 \text{ or } C_2$

where $C^* = \{C_i' | C_i' \in C'\}$

iv)' $\underline{L_2(y,R)}$

$\equiv (|y \wedge R| \neq 1)$

$\equiv ((1001) \wedge (1001) \neq 1) \equiv \underline{\text{true}}$

v)' $\underline{L_1(y,D)}$

$\equiv (|y \wedge D| = 1)$

$\equiv (|(1001) \wedge (1110)| = 1) \equiv \underline{\text{true}}$

Thus all conditions may be satisfied for the triads (t_1,C_3,C_1) and (t_1,C_3,C_2). The effect of the r-function with the second triad of arguments is to change the classification into the classification C again. This confirms a predicted decrease in the match function after the application of this perturbation.

The construction of the limiting classification is achieved by a sequence of steps analogous to the one given above. Each document is matched with each request, and appropriate perturbations are applied for each such document-request pair.

8. Conclusions, Results and Summary

A preliminary program has been written in FORTRAN to implement the assign and remove functions. An experiment with a small set of artificially constructed data indicated that the classes of the limiting classification tended to consist of strongly similar terms. Although the experiment was a small one, it was seen that the assign and remove functions successfully provided the basis for an optimizing strategy.

Optimality was assumed to be achievable by a series of locally optimal steps. A limiting classification constructed in this way must be tested for global optimality by altering it randomly several times to avoid local optima. The construction of limiting classifications is one of optimizing a prescribed function over subsets of vertices of a hypercube.

The introduction of limiting classifications and pseudo-classifications has been valuable in providing a means of conceptualizing and examining a number of basic assumptions in Information Retrieval. Further experiments with larger sets of data are planned.

ACKNOWLEDGEMENT

This work was supported in part by National Science Foundation Grant GN-27224.

REFERENCES

(1) Cleverdon, C.W. and Keen, M. Factors determining the performance of indexing systems, ASLIB Cranfield Research Project, 1966, Vols. 1 and 2.

(2) Doyle, L.B. Semantic road maps for literature searchers, Journal of the ACM 8, 1961, pp. 553-578.

(3) Doyle, L.B. Indexing and abstracting by association, American Documentation 13, 1962, pp. 378-390.

(4) Evans, T.G. A heuristic program to solve geometric-analogy problems, American Federation of Information Processing Societies, AFIPS Conference Proceedings, Vol. 25; 1964 Spring Joint Computer Conference, Washington, April 1964, pp. 327-338.

(5) Gleason, A.M. A search problem in the n-cube, Proceedings of Symposia in Applied Mathematics 10, Combinatorial Analysis, 1960, pp. 175-178.

(6) Hartigan, J.A. Representation of similarity matrices by trees, Journal of the American Statistical Association 62, 1967, pp. 1140-1158.

(7) Jackson, D.M. A note on set of functions for information retrieval, Information Storage and Retrieval 5, 1969, pp. 27-41.

(8) Jackson, D.M. Classification, relevance and information retrieval, in Advances in Computers, Vol. 11, M.C. Yovits, Ed., Academic Press, New York, (forthcoming).

(9) Jackson, D.M. Basis for an improvability measures in information retrieval, in American Society for Information Science Annual Meeting, 32nd, San Francisco, October 1-4, 1969, Proceedings, Vol. 6, pp. 487-495.

(10) Nagy, G. State of the art in pattern recognition, Proceedings of the IEEE 56, 1968, pp. 836-863.

(11) Resnick, A. and Savage, T.T. The consistency of human judgements of

relevance, American Documentation 15, 1964, pp. 93-95.

(12) Salton, G. Computer evaluation for indexing and text processing, Journal of the ACM 15, 1968, pp. 8-36.

(13) Salton, G. and Lesk M.E. The SMART automatic document retrieval system - an illustration, Communications of the ACM 8, 1965, pp. 391-398.

(14) Salton, G. and Lesk, M.E. Relevance assessments and retrieval system evaluation, Information Storage and Retrieval 4, 1969, pp. 343-359.

(15) Sparck Jones, K. and Jackson, D.M. Current approaches to classification and clump-finding at the CLRU, Computer Journal 10, 1967, pp. 29-37.

(16) Swets, J.A. Effectiveness of information retrieval methods, Report No. 1499, Bolt, Beranek and Newman, Cambridge, Mass, 1967.

(17) Taube, M. A note on the pseudo-mathematics of relevance, American Documentation 16, 1965, pp. 69-72.

INTRODUCTION TO THE KEY WORD IN CONTEXT INDEX (KWIC) TO THE ACM IS & R SYMPOSIUM

The editors believe that it is fitting that proceedings devoted to the use of computers in information storage and retrieval use automatic techniques to index the proceedings. Since the papers were typed by the authors, the expense would have been prohibitive to place all of the papers in machine readable form. We took the next best approach, namely, to place in machine readable form the titles and authors of the papers that appear in these proceedings, and the citations that appear with each paper.

A Key Word In Context Index (KWIC) was generated for this set of titles. Every cited paper is referenced back to the article in these proceedings in which the paper was cited. If the paper is multiple cited, it appears only once and is referenced back to all citing papers. Each bibliographic entry in the second portion of the index contains the reference to the citing articles enclosed in parentheses. Thus, (REF MKRS 71) would mean that the reference was cited by the Minker/ Rosenfeld paper. The first part of the index is the conventional sort on the key words in the titles.

We have noted in the introduction to the proceedings that the first KWIC index program was implemented in the late 1950's by Luhn, who first used the term, and by a group at the Rocketdyne Corporation. We hope that this somewhat different application of a KWIC index has served to enhance the proceedings

The editors are indebted to Mr. James Albus of the National Aeronautics and Space Administration for his assistance in keypunching the titles, and to Mr. Jeffry Yeh of the University of Maryland for his efforts in obtaining the KWIC index in the face of numerous misadventures on the computer and a tight time schedule. Without their help the index would never have been developed in time to appear in these proceedings.

Preceding page blank

****KWIC INDEX****

YOJAF59		ABSTRACT	FORMULATION OF DATA PROCESSING PROBLEMS (REF-TEDAR71).=
CHYAG67	BRE,) ENGLISH	ABSTRACT	(REF-CHYFT71).=AUTOMATIC GRAMMATICAL ANALYSIS OF THE HEBREW VERB (IN HE
DOLIA62	INDEXING AND	ABSTRACTING	BY ASSOCIATION (REF-JADOC71).=
KAAEU71	ION OF LIMITED	ACCESS	ARCHIVAL STORAGE IN A TIME SHARED ENVIRONMENT.=EFFICIENT UTILIZAT
KIGEL69	OMPUTER DIRECT	ACCESS	STORAGE MANIPULATION (REF-HUEHM71).=EXPERIMENTS IN LARGE SCALE C
GHSFO69	TION OF RANDOM	ACCESS	INDEX POINTS FOR SEQUENTIAL FILES (REF-WACGE71) (REF-KNGHT71).=FILE ORGANIZATI
BEJOA69	ON-LINE	ACCESS	TO INFORMATION: NSF AS AN AID TO THE INDEXER/CATALOGER (REF-CHYFT71).=
NACAD67		ACCURATELY	DEFINED SYSTEMS (REF-TEDAR71).=
MKRS71	S FOR THE 1971	ACM	INFORMATION STORAGE AND RETRIEVAL SYMPOSIUM .=INTRODUCTION AND PERSPECTIVE
OVCIN70	REX SEMIANNUAL	ACTIVITY	REPORT (REF-MUJDC71).=PROJECT INT
INTRX70	REX SEMIANNUAL	ACTIVITY	REPORT (REF-KAAEU71) .=PROJECT INT
TOJAA66	NALYSIS OF THE	ADDRESS	ASSIGNMENT PROBLEM FOR CLUSTERED KEYS (REF-GURER71).=A
IAMNU66	STRIBUTION AND	ADDRESS	CALCULATION SORTING (REF-GURER71).=NON-UNIFORM KEY DI
HAMAC63	EORY TO A FILE	ADDRESS	PROBLEM (REF-GURER71).=AN APPLICATION OF CODING TH
PEWAR57		ADDRESSING	FOR RANDOM-ACCESS STORAGE (REF-GURER71).=
BOWFO63	GANIZATION AND	ADDRESSING	(REF-WACGE71).=FILE OR
KUJIR70	ON THE	ADMISSIBILITY	OF INPUT QUERIES (REF-DIRRU71).=INTERROGATING A RELATIONAL DATA FILE: REMARKS
KUJIR70	ON THE	ADMISSIBILITY	OF INPUT QUERIES (REF-KUJQQ71).=INTERROGATING A RELATIONAL DATA FILE: REMARKS
AFICP69		AFIPS	1969 SPRING JOINT COMPUTER CONFERENCE (REF-MKRS71).=
BEJOA69	ION: NSF AS AN	AID	TO THE INDEXER/CATALOGER (REF-CHYFT71).=ON-LINE ACCESS TO INFORMAT
KIPSA69	TION: COMPUTER	AIDED	DATA DIRECTORY DEFINITION (REF-TEDAR71).=SYSTEMS ANALYSIS DOCUMENTA
OLYTD68	COMPUTATIONAL	AIDS	FOR OBTAINING A FORMAL DESCRIPTION OF ENGLISH (REF-PAMFS71).=TOWARD THE DEVELO
COUIA62	AN INFORMATION	ALGEBRA	PHASE I REPORT (REF-TEDAR71).=AN INF
BOUQA64	OR HIGH SCHOOL	ALGEBRA	WORK PROBLEMS (REF-KUJQQ71) (REF-SUSMS71) (REF-LOUCC71).=A QUESTION-ANSWERING
FEJAB69	AN	ALGOL-BASED	ASSOCIATE PROCESSOR (REF-KECCN71).=
KNGBT71	AND RETRIEVAL	ALGORITHM	=A BALANCED TREE STORAGE
COKES69	WITH A SEARCH	ALGORITHM	FOR THE DATA BASE OF A HUMAN BELIEF SYSTEM (REF-SUSMS71) (REF-PAMFS71) .=EXPER
ADGAO	AN	ALGORITHM	FOR THE ORGANIZATION OF INFORMATION (REF-KNGBT71).=
EVUDS71	STRUCTURES AND	ALGORITHMIC	DETERMINACY (REF-WIAFS71).=DATA
KNUAC68	MENTAL	ALGORITHMS	(REF-KNGBT71).=THE ART OF COMPUTER PROGRAMMING, VOL 1: FUNDA
DOBHF71	OLVE SYNTACTIC	AMBIGUITY	=HOW FEATURES RES
KECPA70	PROPOSED	AMERICAN	NATIONAL STANDARD IDENTIFICATION OF STATES.....FOR INFORMATION INTERCHANGE (
THJPS69	THE SYNTACTIC	ANALYSIS	OF ENGLISH SENTENCES (REF-PAMFS71).=A PROGRAM FOR
TOJAA66		ANALYSIS	OF THE ADDRESS ASSIGNMENT PROBLEM FOR CLUSTERED KEYS (REF-GURER71).=
WAUCL69	EM FOR TEXTUAL	ANALYSIS	(REF-SUSMS71).=COMPUTATIONAL LINGUISTIC TECHNIQUES IN AN ON-LINE SYST
WOWTN70	TURAL LANGUAGE	ANALYSIS	(REF-KECCN71) (REF-DOBHF71) .=TRANSITION NETWORK GRAMMARS FOR NA
MECAI67	THE	ANALYSIS	OF INFORMATION SYSTEMS (REF-GURER71).=
SIRLA68	LINGUISTIC	ANALYSIS	OF CONSTRUCTED RESPONSES IN CAI (REF-PAMFS71).=
LABTA66	THEORETICAL	ANALYSIS	OF INFORMATION SYSTEMS (REF-TEDAR71).=
KUSAS67	ATIC SYNTACTIC	ANALYSIS	(REF-PAMFS71).=AUTOM
KIPSA69	SYSTEMS	ANALYSIS	DOCUMENTATION: COMPUTER AIDED DATA DIRECTORY DEFINITION (REF-TEDAR71).=
FLIAI61		ANALYSIS	OF INTERNAL COMPUTER SORTING (REF-WIAFS71).=
GAPPT67	DY IN SEMANTIC	ANALYSIS	(REF-PAMFS71).=PREDICATION TYPING: A PILOT STU
BENAQ63	AN	ANALYSIS	OF QUESTIONS: PRELIMINARY REPORT (REF-KUJQQ71).=
CHYAG67	IC GRAMMATICAL	ANALYSIS	OF THE HEBREW VERB (IN HEBREW) ENGLISH ABSTRACT (REF-CHYFT71).=AUTOMAT
CHYAG66	IC GRAMMATICAL	ANALYSIS	OF THE HEBREW VERB (IN HEBREW) (REF-CHYFT71).=AUTOMAT
CHPPM68	MORPHOLOGICAL	ANALYSIS	(REF-PAMFS71).=A PROCEDURE FOR
AQLNA65	S. PART I,	ANALYSIS	(REF-KUJQQ71).=A NEW APPROACH TO THE LOGICAL THEORY OF INTERROGATIVE
CHTGI68	OF REDUCTIONS	ANALYSIS	PROGRAMS FOR CONTEXT FREE GRAMMAR (REF-LODCC71).=ON THE GENERATION AND IMPLEME
KUSMP63	PATH SYNTACTIC	ANALYZER	(REF-LOUCC71).=MULTIPLE-
SHSNS69	IONAL QUESTION	ANSWERER	DESCRIPTION AND EXAMPLES (REF-SUSMS71).=A NET STRUCTURE BASED RELAT
SIRAE65		ANSWERING	ENGLISH QUESTIONS BY COMPUTER: A SURVEY (REF-MKRS71) (REF-PAMFS71) (REF-SUSMS7
SIRNL70	GUAGE QUESTION	ANSWERING	SYSTEMS (REF-PAMFS71).=NATURAL LAN

```
KUJAQ67                          ANSWERING    QUESTIONS BY COMPUTER: A LOGICAL STUDY (REF-KUJQQ71).=
KUJAQ67                          ANSWERING    QUESTIONS BY COMPUTER: A LOGICAL STUDY (REF-DIRRD71).=
KIGAL70                          APL/B5500     THE LANGUAGE AND ITS IMPLEMENTATION (REF-HUEHM71).=
IVKAU68                          APL/360       USERS MANUAL (REF-HUEHM71).=
WRMML60          A LARGE INDEX;  APPENDIX      THE SOUNDEX CODE (REF-MUJDC71).=MECHANIZING
HAMAC63                    AN    APPLICATION  OF CODING THEORY TO A FILE ADDRESS PROBLEM (REF-GURER71).=
DAOE65           SYSTEM : AN     APPLICATION  OF INTEGER PROGRAMMING (REF-WACQE71).=ON OPTIMAL EXTRACING FROM A MULTIPLE FIL
GHSAF68                          APPLICATION  OF FINITE GEOMETRY IN FILE ORGANIZATION FOR RECORDS WITH MULTIPLE-VALUED ATTRI
GRCAT69                          APPLICATION  OF THEOREM PROVING TO PROBLEM SOLVING (REF-SUSMS71).=
KOAOD66          DISCIPLINE AND  APPLICATIONS  (REF-GURER71).=AN OCCUPANCY
CARIS58           LOGIC AND ITS  APPLICATIONS  (REF-KUJQQ71).=INTRODUCTION TO SYMBOLIC
FEWIP68          THEORY AND ITS  APPLICATIONS  (REF-GURER71).=AN INTRODUCTION TO PROBABILITY
FRAMR68          RIEVAL METHODS  APPLIED       TO JEWISH CASE LAW (REF-CHYFT71).=MODERN RET
TEDAR71                    AN    APPROACH     TO RESEARCH IN FILE ORGANIZATION.=
AQLNA65                A NEW     APPROACH     TO THE LOGICAL THEORY OF INTERROGATIVES, PART I, ANALYSIS (REF-KUJQQ71).=
SPKCA67              CURRENT     APPROACHES   TO CLASSIFICATION AND CLUMP-FINDING AT THE CLRU (REF-JADOC71).=
LASA63                 SOME      APPROACHES   TO THE THEORY OF INFORMATION SYSTEMS (REF-WACQE71).=
KAAEU71          LIMITED ACCESS  ARCHIVAL      STORAGE IN A TIME SHARED ENVIRONMENT.=EFFICIENT UTILIZATION OF
NAGSA68            STATE OF THE  ART           IN PATTERN RECOGNITION (REF-JADOC71).=
STMAI70          A STATE OF THE  ART           REPORT (REF-MKRS71).=AUTOMATIC INDEXING:
KNDAC68                    THE   ART           OF COMPUTER PROGRAMMING, VOL 1: FUNDAMENTAL ALGORITHMS (REF-KNGBT71).=
KNRAC69                    THE   ART           OF COMPUTER PROGRAMMING,VOL. 1 (REF-HUEHM71).=
KUJLA69                LOGICAL   ASPECTS       OF QUESTION-ANSWERING BY COMPUTER (REF-KUJQQ71).=
CHNAT65                          ASPECTS       OF THE THEORY OF SYNTAX (REF-LODCC71) (REF-DOBHF71) (REF-PAMFS71) .=
SAGRA69                RELEVANCE ASSESSMENTS   AND RETRIEVAL SYSTEM EVALUATION (REF-JADOC71).=
TOJAA66          OF THE ADDRESS  ASSIGNMENT    PROBLEM FOR CLUSTERED KEYS (REF-GURER71).=ANALYSIS
FEJAB69          AN ALGOL-BASED  ASSOCIATE     PROCESSOR (REF-KECCN71).=AN ALG
SGHSA64             STATISTICAL  ASSOCIATION   METHODS FOR MACHANIZED DOCUMENTATION, (REF-MKRS71) .=
DOLIA62          ABSTRACTING BY  ASSOCIATION    (REF-JADOC71).=INDEXING AND
SUSMS71          TIC DATA IN AN  ASSOCIATIVE   NET.=MANAGING SEMAN
KIJMA61              A MAGNETIC  ASSOCIATIVE   MEMORY (REF-GURER71).=
CHBSH63          IZABLE FORM OF  ASSOCIATIVE   MEMORY (REF-GURER71).=SHIEF: A REAL
ASWTR68          N INTERPRETIVE  ASSOCIATIVE   PROCESSOR WITH DEDUCTIVE CAPABILITIES (REF-KECCN71).=TRAMP: A
NIGSP69          JECT:SEARCHING  ATOMIC        ENERGY LAW BY COMPUTER (REF-CHYFT71).=THE STATUS PRO
BOUAM65                    THE   ATOMIZATION   OF MEANING (REF-PAMFS71).=
GHSAF68          ULTIPLE-VALUED  ATTRIBUTES     (REF-GURER71).=APPLICATION OF FINITE GEOMETRY IN FILE ORGANIZATION FOR RECORD
PAMFS71          F SEMANTICS IN  AUTOMATED     LANGUAGE PROCESSING.=THE FUNCTION O
MOCAL69                          AUTOMATED     LANGUAGE PROCESSING (REF-PAMFS71).=
SAGAL68                          AUTOMATED     LANGUAGE PROCESSING (REF-PAMFS71).=
MOCLA69          INGUISTICS AND  AUTOMATED     LANGUAGE PROCESSING (REF-PAMFS71).=L
IBMTA68                THE TIME  AUTOMATED     GRID SYSTEM (TAG): SALES AND SYSTEMS GUIDE (REF-TEDAR71).=
BOUAL67                          AUTOMATED     LANGUAGE PROCESSING (REF-PAMFS71).=
CUPAR69                    AN    AUTOMATED     RECOGNTION GRAMMAR FOR ENGLISH (REF-LODCC71).=
SAGAI63                          AUTOMATIC     INFORMATION ORGANIZATION AND RETRIEVAL (REF-MKRS71) (REF-GURER71).=
SAGDM66          NG PROBLEMS IN  AUTOMATIC      INFORMATION RETRIEVAL (REF-GUREP71).=DATA MANIPULATION AND PROGRAMMI
OTMIL69           LANGUAGES FOR  AUTOMATIC     LANGUAGES PROCESSING (REF-PAMFS71).=INTERMEDIATE
STMAI70                          AUTOMATIC     INDEXING: A STATE OF THE ART REPORT (REF-MKRS71).=
MABAD67                          AUTOMATIC     DATA COMPRESSION (REF-MUJDC71).=
PRAAP69                          AUTOMATIC     PROCESSING OF MEDICAL ENGLISH (REF-PAMFS71).=
SAGSA65                THE SMART AUTOMATIC     DOCUMENT RETRIEVAL SYSTEM -- AN ILLUSTRATION (REF-JADOC71).=
MCWTA69                TOWARD THE AUTOMATIC     DESIGN OF DATA ORGANIZATION FOR LARGE SCALE INFORMATION PROCESSING SYSTEMS (RE
KUSAS67                          AUTOMATIC     SYNTACTIC ANALYSIS (REF-PAMFS71).=
KOMAQ69                          AUTOMATIC     QUESTION-ANSWERING OF ENGLISH-LIKE QUESTIONS ABOUT SIMPLE DIAGRAMS (REF-SUSMS7
GWCLB61          BASEBALL: AN    AUTOMATIC     QUESTION-ANSWERER, (REF-MKRS71).=
EALAD67                          AUTOMATIC     DETERMINATION OF PARTS OF SPEECH OF ENGLISH WORDS (REF-PAMFS71).=
CHYAG66                          AUTOMATIC     GRAMMATICAL ANALYSIS OF THE HEBREW VERB ( IN HEBREW) (REF-CHYFT71).=
```

CHYAG67		AUTOMATIC	GRAMMATICAL ANALYSIS OF THE HEBREW VERB (IN HEBREW) ENGLISH ABSTRACT (REF-CHY
OVLLA69	LIBRARY	AUTOMATION	A CRITICAL REVIEW (REF-SUSMS71).=
KIGLC63		AUTOMATION	AND THE LIBRARY OF CONGRESS (REF-MKRS71).=
FRACI69	N RETRIEVAL AS	AUXILIARY	TOOLS IN HEBREW LEGAL RESEARCH (IN HEBREW) (REF-CHYFT71).=THE COMPUTER AND I
FOCIS65	ETRIEVAL USING	AVL	TREES (REF-KNGBT71).=INFORMATION STORAGE AND R
GOSBP65		BACKTRACK	PROGRAMMING (REF-GURER71).=
KNGBT71	A	BALANCED	TREE STORAGE AND RETRIEVAL ALGORITHM.=
FEUBT68		BALANCED	TREE SEARCHING (REF-KNGBT71).=
WOWSI67	TRUCTURED DATA	BASE	(REF-PAMFS71).=SEMANTIC INTERPRETATION OF ENGLISH QUESTIONS ON A S
COUSG69	NERALIZED DATA	BASE	MANAGEMENT SYSTEMS (REF-MKRS71) .=A SURVEY OF GE
COJEM69	N ON-LINE DATA	BASE	UNDER TSS/360 (REF-KAAEU71).=ESTABLISHMENT AND MAINTENANCE OF A STORAGE HIERAR
COKES69	M FOR THE DATA	BASE	OF A HUMAN BELIEF SYSTEM (REF-SUSMS71) (REF-PAMFS71) .=EXPERIMENTS WITH A SEAR
GWCLB61		BASEBALL	AN AUTOMATIC QUESTION-ANSWERER. (REF-MKRS71).=
SUSST69	EMANTIC THEORY	BASED	UPON INTERACTIVE MEANING (REF-SUSMS71).=A S
SHSNS69	NET STRUCTURE	BASED	RELATIONAL QUESTION ANSWERER: DESCRIPTION AND EXAMPLES (REF-SUSMS71).=A
JAUBI69		BASIS	FOR AN IMPROVABILITY MEASURE IN INFORMATION RETRIEVAL (REF-JAUOC71).=
COKES69	SE OF A HUMAN	BELIEF	SYSTEM (REF-SUSMS71) (REF-PAMFS71) .=EXPERIMENTS WITH A SEARCH ALGORITHM FOR T
WEJEC66	COMMUNICATION	BETWEEN	MAN AND MACHINE (REF-LODCC71).=ELIZA:A COMPUTER PROGRAM FOR THE STUDY OF NATUR
BRWGP66	L OF	BIBLIOGRAPHIC	INFORMATION AT THE NASA SCIENTIFIC AND TECHNICAL INFORMATION FACILITY (REF-SUS
KNUOB70	OPTIMUM	BINARY	SEARCH TREES (REF-KNGBT71).=
EAJGR65	COGNIZER FOR A	BNF	GRAMMAR (REF-DOBHF71).=GENERATING A RE
HUEHM71	KIT METHOD FOR	BUILDING	DATA MANAGEMENT PROGRAMS.=A HEATH
FIJUG70	SYSTEM ON THE	B5500	COMPUTER (REF-HUEHM71).=A USERS GUIDE TO THE OPERATION OF AN INFORMATION STORA
SIRLA68	D RESPONSES IN	CAI	(REF-PAMFS71).=LINGUISTIC ANALYSIS OF CONSTRUCTE
TAMNU66	ON AND ADDRESS	CALCULATION	SORTING (REF-GURER71).=NON-UNIFORM KEY DISTRIBUTI
TAACR41	ON THE	CALCULUS	OF RELATIONS (REF-KUJOQ71).=
CHECS69	ER SOLUTION OF	CALCULUS	WORD PROBLEMS (REF-SUSMS71).=COMPUT
LOEDC68	DATA CODE FOR	CALENDAR	DATE FOR MACHINE-TO-MACHINE DATA INTERCHANGE (REF-MUJDC71).=
ASWTR68	ITH DEDUCTIVE	CAPABILITIES	(REF-KECCN71).=TRAMP: AN INTERPRETIVE ASSOCIATIVE PROCESSOR W
WIRCR62	R RETRIEVAL OF	CASE	LAW (REF-CHYFT71).=COMPUTE
SCACS69	D RETRIEVAL OF	CASE	LAW WITHOUT INDEXING: THE HEBREW RESPONSA PROJECT (REF-CHYFT71).=COMPUTERIZED
FIC CC68	THE	CASE	FOR CASE (REF-PAMFS71).=
FRAMR68	LIED TO JEWISH	CASE	LAW (REF-CHYFT71).=MODERN RETRIEVAL METHODS APP
FIC CC68	THE CASE FOR	CASE	(REF-PAMFS71).=
FICTM66	DERN THEORY OF	CASE	(REF-PAMFS71).=TOWARD A MO
ACWSC54	SOLVABLE	CASES	OF THE DECISION PROBLEM (REF-DIRRD71).=
CHBCT65	FOR RETRIEVAL	CENTER	OPERATIONS (REF-GURER71).=COLLOQUIUM ON TECHNICAL PRECONDITIONS
KAMLP62	SEARCHING	CENTERS	(REF-CHYFT71).=A LONG-RANGE PROGRAM FOR MECHANIZED LEGAL AND PATENT
HITSC62	PROPERTIES OF	CERTAIN	TREES (REF-KNGBT71).=SOME COMBINATORIAL
PRFSU39	Y OF CODES AND	CIPHERS	(REF-MUJDC71) .=SECRET AND URGENT, THE STOR
DIRRU69	ROBLEM FOR THE	CLASS	OF DEFINITE FORMULAS (REF-DIRRD71).=THE RECURSIVE UNSOLVABILITY OF THE DECISIO
DIRRD71	BLEM FOR	CLASSES	OF PROPER FORMULAS.=THE RELATIONAL DATA FILE AND THE DECISION PRO
JAUOC71	OPTIMAL	CLASSIFICATION	AND ITS CONSEQUENCES.=
SPKCA67	APPROACHES TO	CLASSIFICATION	AND CLUMP-FINDING AT THE CLRU (REF-JAUOC71).=CURRENT
PAMHC67	HOMOGRAPHS II:	CLASSIFICATION	AND IDENTIFICATION (REF-PAMFS71).=HOMOGR
PAMHC63	HOMOGRAPHS I:	CLASSIFICATION	AND IDENTIFICATION (REF-PAMFS71).=
JAUCR		CLASSIFICATION	RELEVANCE AND INFORMATION RETRIEVAL (REF-JAUOC71).=
SPKCA67	NDING AT THE	CLRU	(REF-JAUOC71).=CURRENT APPROACHES TO CLASSIFICATION AND CLUMP-FI
SPKCA67	SIFICATION AND	CLUMP-FINDING	AT THE CLRU (REF-JAUOC71).=CURRENT APPROACHES TO CLAS
TOJAA66	NT PROBLEM FOR	CLUSTERED	KEYS (REF-GURER71).=ANALYSIS OF THE ADDRESS ASSIGNME
VAPCC70	CAN	COBOL	COPE (REF-TEDAR71).=
WRMML60	X: THE SOUNDEX	CODE	(REF-MUJDC71).=MECHANIZING A LARGE INDEX; APPENDI
LOEDC68	DATA	CODE	FOR CALENDAR DATE FOR MACHINE-TO-MACHINE DATA INTERCHANGE (REF-MUJDC71).=
KAJLG68	R GEOGRAPHICAL	CODE	FOR MANAGEMENT INFORMATION SYSTEMS (REF-MUJDC71).=A LINEA
KAUTC67	THE	CODEBREAKERS	(REF-MUJDC71).=

```
VEJED      AND CORRECTING    CODES          FOR THE DECIMAL NUMBER SYSTEM (REF-MUJDC71).=ERROR DETECTING
NEPEE61    ARIABLE-LENGTH    CODES           (REF-MUJDC71).=EFFICIENT ERROR-LIMITING V
PRFSU39    , THE STORY OF    CODES          AND CIPHERS (REF-MUJDC71) .=SECRET AND URGENT
HUDMC      MUM REDUNDANCY    CODES           (REF-KAAEU71).=A METHOD FOR THE CONSTRUCTION OF MINI
NUWCW68    MPRESSION WORD    CODING         TECHNIQUES FOR INFORMATION RETRIEVAL (REF-MUJDC71).=CO
SCELE67    OR COMPRESSION    CODING          (REF-MUJDC71).=A LANGUAGE ELEMENT F
MAWFC68     USING HUFFMAN    CODING          (REF-MUJDC71).=FILE COMPRESSION
KULCG63                      CODING         OF GRAMMATICAL DATA (REF-MUJDC71).=
HAMAC63    APPLICATION OF    CODING         THEORY TO A FILE ADDRESS PROBLEM (REF-GURER71).=AN
CHBCT65                      COLLOQUIUM     ON TECHNICAL PRECONDITIONS FOR RETRIEVAL CENTER OPERATIONS (REF-GURER71).=
GURER71    THE RANDOMIZED    COMBINATORIAL  FILE STRUCTURE.=ELEMENTS OF
HITSC62            SOME      COMBINATORIAL  PROPERTIES OF CERTAIN TREES (REF-KNGBT71).=
GSRRC69      A RANDOMIZED    COMBINATORIAL  FILE STRUCTURE FOR STORAGE AND RETRIEVAL SYSTEMS (REF-GURER71).=
LEUMT64    CHINE TOOLS OF    COMBINATORICS   (REF-GURER71).=THE MA
LUVMR70    RETRIEVAL WITH    COMBINED       INDEXES (REF-GURER71).=MULTI-ATTRIBUTE
YNCMI62                      COMIT          AS AN IR LANGUAGE (REF-MKRS71).=
MUJDC71    SSING OF LARGE    COMMERCIAL     FILES.=DATA COMPRESSION TECHNIQUES FOR ECONOMIC PROCE
RONUE68    OF ENGLISH FOR    COMMUNICATING  WITH A COMPUTER (REF-LODCC71).=THE USE
WEJEC66    URAL LANGUAGE     COMMUNICATION  BETWEEN MAN AND MACHINE (REF-LODCC71).=ELIZA:A COMPUTER PROGRAM FOR THE STUDY
STISK63    HINE GRAPHICAL    COMMUNICATION  SYSTEM (REF-HUEHM71) .=SKETCHPAD, A MAN-MAC
HAUCL63                      COMMUNICATION   A LOGICAL MODEL (REF-KUJQQ71).=
MECMM70       MAN-MACHINE    COMMUNICATIONS  (REF-GURER71).=
SNMMV70    IRTUES OF TEXT    COMPACTION      (REF-MUJDC71) .=THE MYRIAD V
WIAMC70    ETHODOLOGY FOR    COMPARISON     OF FILE ORGANIZATION AND PROCESSING PROCEDURES FOR HIERARCHICAL STORAGE STRUCT
CHACF69              A       COMPARISON     OF FILE ORGANIZATION TECHNIQUES (REF-WIAFS71).=
KECNL68    TURAL LANGUAGE    COMPILER       FOR ONLINE DATA MANAGEMENT (REF-KECCN71).=A NA
KECNL68    TURAL LANGUAGE    COMPILER       FOR ON-LINE DATA MANAGEMENT (REF-PAMFS71).=A NA
KEGNL68    TURAL LANGUAGE    COMPILER       FOR ON-LINE DATA MANAGEMENT (REF-SUSMS71) .=A NA
GURTL69    HABLE LANGUAGE    COMPREHENDER    (REF-PAMFS71) (REF-SUSMS71) .=THE TEAC
KEWET68    RANSMISSION OF    COMPRESSED     DATA (REF-MUJDC71).=ERRORS IN T
MUJDC71            DATA      COMPRESSION    TECHNIQUES FOR ECONOMIC PROCESSING OF LARGE COMMERCIAL FILES.=
MAWFC68            FILE      COMPRESSION    USING HUFFMAN CODING (REF-MUJDC71).=
NUWCW68                      COMPRESSION    WORD CODING TECHNIQUES FOR INFORMATION RETRIEVAL (REF-MUJDC71).=
SCELE67    GE ELEMENT FOR    COMPRESSION    CODING (REF-MUJDC71).=A LANGUA
MABAD67    AUTOMATIC DATA    COMPRESSION     (REF-MUJDC71).=AUTOMA
DEPCC69          THE COPAK   COMPRESSOR      (REF-MUJDC71).=
SIRCM68              A       COMPUTATIONAL  MODEL OF VERBAL UNDERSTANDING (REF-SUSMS71).=
WAUCL69                      COMPUTATIONAL  LINGUISTIC TECHNIQUES IN AN ON-LINE SYSTEM FOR TEXTUAL ANALYSIS (REF-SUSMS71).
PAMCM67                      COMPUTATIONAL  MORPHOLOGY (REF-PAMFS71).=
SCRTC69          TOWARDS A   COMPUTATIONAL  FORMALIZATION OF NATURAL LANGUAGE SEMANTICS (REF-PAMFS71).=
SIRCM68              A       COMPUTATIONAL  MODEL OF VERBAL UNDERSTANDING (REF-PAMFS71).=
OLYTD68    DEVELOPMENT OF    COMPUTATIONAL  AIDS FOR OBTAINING A FORMAL DESCRIPTION OF ENGLISH (REF-PAMFS71).=TOWARD THE
HAUIC67    NTRODUCTION TO    COMPUTATIONAL  LINGUISTICS (REF-GURER71).=I
BAYOC      HE OUTLOOK FOR    COMPUTATIONAL  SEMANTICS (REF-PAMFS71).=T
WIRCR62                      COMPUTER       RETRIEVAL OF CASE LAW (REF-CHYFT71).=
WEJEC66            ELIZA:A   COMPUTER       PROGRAM FOR THE STUDY OF NATURAL LANGUAGE COMMUNICATION BETWEEN MAN AND MACHIN
SIRAE65    H QUESTIONS BY    COMPUTER        A SURVEY (REF-MKRS71) (REF-PAMFS71) (REF-SUSMS71) .=ANSWERING ENGLIS
MAJDR67    N OF REAL-TIME    COMPUTER       SYSTEMS (REF-TEDAR71).=DESIG
NIGSP69     ENERGY LAW BY    COMPUTER        (REF-CHYFT71).=THE STATUS PROJECT:SEARCHING ATOMIC
LERCS67              A       COMPUTER       SYSTEM FOR INFERENCE EXECUTION AND DATA RETRIEVAL (REF-KUJQQ71).=
SAGCE68                      COMPUTER       EVALUATION FOR INDEXING AND TEXT PROCESSING (REF-JADOC71).=
RONUE68    ICATING WITH A    COMPUTER        (REF-LODCC71).=THE USE OF ENGLISH FOR COMMUN
KAADR67    A-RETRIEVAL BY    COMPUTER        A CRITICAL SURVEY (REF-SUSMS71) (REF-KUJQQ71).=DAT
KIPSA69    DOCUMENTATION:    COMPUTER       AIDED DATA DIRECTORY DEFINITION (REF-TEDAR71).=SYSTEMS ANALYSIS
KIGEL69    IN LARGE SCALE    COMPUTER       DIRECT ACCESS STORAGE MANIPULATION (REF-HUEHM71).=EXPERIMENTS
KNDAC68        THE ART OF    COMPUTER       PROGRAMMING, VOL 1: FUNDAMENTAL ALGORITHMS (REF-KNGBT71).=
KUJAQ67    G QUESTIONS BY    COMPUTER        A LOGICAL STUDY (REF-DIRRD71).=ANSWERIN
```

KUJAQ67	G QUESTIONS BY	COMPUTER	A LOGICAL STUDY (REF-KUJQQ71).=ANSWERIN
HUECS70	THE	COMPUTER	SCIENCE TEACHING LABORATORY AT THE UNIVERSITY OF WASHINGTON (REF-HUEHM71).=
KNRAC69	THE ART OF	COMPUTER	PROGRAMMING VOL. 1 (REF-HUEHM71).=
LERCS67	A	COMPUTER	SYSTEM FOR INFERENCE EXECUTION AND DATA RETRIEVAL (REF-DIRRD71).=
KUJLA69	N-ANSWERING BY	COMPUTER	(REF-KUJQQ71).=LOGICAL ASPECTS OF QUESTIO
HOJSS62	ATUTORY LAW BY	COMPUTER	(REF-CHYFT71).=SEARCHING ST
CHECS69		COMPUTER	SOLUTION OF CALCULUS WORD PROBLEMS (REF-SUSMS71).=
FIJUG70	M ON THE B5500	COMPUTER	(REF-HUEHM71).=A USERS GUIDE TO THE OPERATION OF AN INFORMATION STORAGE AND R
FLIAI61	IS OF INTERNAL	COMPUTER	SORTING (REF-WIAFS71).=ANALYS
ELW CT63	THE	COMPUTER	AS A TOOL FOR LEGAL RESEARCH (REF-CHYFT71).=
GAPNL63	NGUAGE AND THE	COMPUTER	(REF-GURER71).=NATURAL LA
FRACI69	THE	COMPUTER	AND INFORMATION RETRIEVAL AS AUXILIARY TOOLS IN HEBREW LEGAL RESEARCH (IN HEB
BOUST68	CTIC THEORY IN	COMPUTER	IMPLEMENTATIONS (REF-PAMFS71).=SYNTA
AFICP69	9 SPRING JOINT	COMPUTER	CONFERENCE (REF-MKRS71).=AFIPS 196
SCACS69		COMPUTERIZED	STORAGE AND RETRIEVAL OF CASE LAW WITHOUT INDEXING: THE HEBREW RESPONSA PROJEC
AFICP69	JOINT COMPUTER	CONFERENCE	(REF-MKRS71).=AFIPS 196º SPRING
OVCCI65	PLANNING	CONFERENCE	ON INFORMATION TRANSFER EXPERIMENTS (INTREX) (REF-MKRS71) .=
SPJIS65	HE SECOND	CONGRESS	(REF-GURER71).=INFORMATION SYSTEMS SCIENCES: PROCEEDING OF T
KIGLC63	THE LIBRARY OF	CONGRESS	(REF-MKRS71).=AUTOMATION AND
JADOC71	CATION AND ITS	CONSEQUENCES	=OPTIMAL CLASSIFI
REACH64	THE	CONSISTENCY	OF HUMAN JUDGEMENTS OF RELEVANCE (REF-JADOC71).=
SIRLA68	IC ANALYSIS OF	CONSTRUCTED	RESPONSES IN CAI (REF-PAMFS71).=LINGUIST
HUDMC	METHOD FOR THE	CONSTRUCTION	OF MINIMUM REDUNDANCY CODES (REF-KAAEU71).=A
BOAEN60	OF DICTIONARY	CONSTRUCTION	(REF-KNGBT71).=ON THE EFFICIENCY OF A NEW METHOD
CHTG168	S PROGRAMS FOR	CONTEXT	FREE GRAMMAR (REF-LODCC71).=ON THE GENERATION AND IMPLEMENTATION OF REDUCTIONS
GRCSI70	TION OF FORMAL	CONTROL	SYSTEMS (REF-TEDAR71).=SPECIFICATION AND INTERROGA
CGMIE58	INFORMATION	CONTROL	(REF-MKRS71).=
KECCN71	THE	CONVERSE	NATURAL LANGUAGE DATA MANAGEMENT SYSTEM: CURRENT STATUS AND PLANS.=
DEPCC69	THE	COPAK	COMPRESSOR (REF-MUJDC71).=
VAPCC70	CAN COBOL	COPE	(REF-TEJAR71).=
ROFIC66	THE IBM	CORE	GRAMMAR OF ENGLISH (REF-LODCC71).=
VEJED	DETECTING AND	CORRECTING	CODES FOR THE DECIMAL NUMBER SYSTEM (REF-MUJDC71).=ERROR
CESMT	ANSLATION: THE	CORRELATIONAL	SOLUTION (REF-PAMFS71).=MECHANICAL TR
CHWTR62	IEVAL REQUESTS	COUCHED	IN A 'SEMIFORMAL' ENGLISH-LIKE LANGUAGE, (REF-MKRS71).=TRANSLATION OF RETR
CVLLA69	Y AUTOMATION:A	CRITICAL	REVIEW (REF-SUSMS71).=LIBRAR
SCG67	N RETRIEVAL: A	CRITICAL	VIEW (REF-GURER71).=INFORMATIO
KAADR67	BY COMPUTER: A	CRITICAL	SURVEY (REF-SUSMS71) (REF-KUJQQ71).=DATA-RETRIEVAL
GAHCR56		CRYPTANALYSIS	(REF-MUJDC71).=
LODCC71		CUE	A CUSTOMIZED SYSTEM FOR RESTRICTED, NATURAL ENGLISH.=
KECCN71	SYSTEM:	CURRENT	STATUS AND PLANS.=THE CONVERSE NATURAL LANGUAGE DATA MANAGEMENT
SPKCA67		CURRENT	APPROACHES TO CLASSIFICATION AND CLUMP-FINDING AT THE CLRU (REF-JADOC71).=
LODCC71	CUE: A	CUSTOMIZED	SYSTEM FOR RESTRICTED, NATURAL ENGLISH.=
WEPCR71		CYLINDERS	A RELATIONAL DATA STRUCTURE (REF-SUSMS71).=
MUJDC71		DATA	COMPRESSION TECHNIQUES FOR ECONOMIC PROCESSING OF LARGE COMMERCIAL FILES.=
TUSD63	SEQUENTIAL	DATA	PROCESSING DESIGN (REF-TEDAR71).=
KECCN71	TURAL LANGUAGE	DATA	MANAGEMENT SYSTEM: CURRENT STATUS AND PLANS.=THE CONVERSE NA
WEPCR71	: A RELATIONAL	DATA	STRUCTURE (REF-SUSMS71).=CYLINDERS
YOJAF59	FORMULATION OF	DATA	PROCESSING PROBLEMS (REF-TEDAR71).=ABSTRACT
WOWSI67	N A STRUCTURED	DATA	BASE (REF-PAMFS71).=SEMANTIC INTERPRETATION OF ENGLISH QUESTIONS O
HUEHM71	D FOR BUILDING	DATA	MANAGEMENT PROGRAMS.=A HEATHKIT METHO
DIRRD71	THE RELATIONAL	DATA	FILE AND THE DECISION PROBLEM FOR CLASSES OF PROPER FORMULAS.=THE RE
SUSMS71	AGING SEMANTIC	DATA	IN AN ASSOCIATIVE NET.=MAN
WCWTA69	ATIC DESIGN OF	DATA	ORGANIZATION FOR LARGE SCALE INFORMATION PROCESSING SYSTEMS (REF-WACQE71).=TOW
MABAD67	AUTOMATIC	DATA	COMPRESSION (REF-MUJDC71).=
MIJRD70	RELATIONAL	DATA	SYSTEM STUDY (REF-MKRS71) .=
MAMRD66	RELATIONAL	DATA	FILE I: DESIGN PHILOSOPHY (REF-DIRRD71).=
MALMF69	LARGE ON-LINE	DATA	FILES (REF-WIAFS71).=A MODEL FOR FILE STRUCTURE DETERMINATION FOR
		DATA	MANIPULATION AND PROGRAMMING PROBLEMS IN AUTOMATIC INFORMATION RETRIEVAL (REF-

MKRGD69	GENERALIZED	DATA	MANAGEMENT SYSTEMS - SOME PRESPECTIVES (REF-MKRS71).=
LERRD65	RELATIONAL	DATA	FILE: A TOOL FOR MECHANIZED INFERENCE EXECUTION AND DATA RETRIEVAL (REF-KUJQQ7
KEGNL68	ER FOR ON-LINE	DATA	MANAGEMENT (REF-SUSMS71) .=A NATURAL LANGUAGE COMPIL
LERTM65	RELATIONAL	DATA	FILE: A TOOL FOR MECHANIZED INFERENCE EXECUTION AND DATA RETRIEVAL (REF-KECCN7
KEWET68	OF COMPRESSED	DATA	(REF-MUJDC71).=ERRORS IN TRANSMISSION
LERCS67	EXECUTION AND	DATA	RETRIEVAL (REF-KUJQQ71).=A COMPUTER SYSTEM FOR INFERENCE
LERCS67	EXECUTION AND	DATA	RETRIEVAL (REF-DIRRD71).=A COMPUTER SYSTEM FOR INFERENCE
KUJIR70	G A RELATIONAL	DATA	FILE: REMARKS ON THE ADMISSIBILITY OF INPUT QUERIES (REF-DIRRD71).=INTERROGATI
LERRD66	RELATIONAL	DATA	FILE II: IMPLEMENTATION (REF-DIRRD71).=
KECNL68	ER FOR ON-LINE	DATA	MANAGEMENT (REF-PAMFS71).=A NATURAL LANGUAGE COMPIL
LERTM65	EXECUTION AND	DATA	RETRIEVAL (REF-KECCN71).=RELATIONAL DATA FILE: A TOOL FOR MECHANIZED INFERENCE
KUJIR70	G A RELATIONAL	DATA	FILE: REMARKS ON THE ADMISSIBILITY OF INPUT QUERIES (REF-KUJQQ71).=INTERROGATI
LOEDC68	INE-TO-MACHINE	DATA	INTERCHANGE (REF-MUJDC71).=DATA CODE FOR CALENDAR DATE FOR MACH
KECNL68	LER FOR ONLINE	DATA	MANAGEMENT (REF-KECCN71).=A NATURAL LANGUAGE COMPI
LOEDC68		DATA	CODE FOR CALENDAR DATE FOR MACHINE-TO-MACHINE DATA INTERCHANGE (REF-MUJDC71).=
KULCG63	OF GRAMMATICAL	DATA	(REF-MUJDC71).=CODING
IBMDM70		DATA	MANAGEMENT FACILITIES OF TSS/360 (REF-KAAEU71).=
LERRD69	RELATIONAL	DATA	FILE: EXPERIENCE WITH A SYSTEM FOR PROPOSITIONAL DATA STORAGE AND INFERENCE EX
LERRD69	PROPOSITIONAL	DATA	STORAGE AND INFERENCE EXEC (REF-DIRRD71).=RELATIONAL DATA FILE: EXPERIENCE WIT
LERRD65	EXECUTION AND	DATA	RETRIEVAL (REF-KUJQQ71).=RELATIONAL DATA FILE: A TOOL FOR MECHANIZED INFERENCE
KECDM68		DATA	MANAGEMENT IN ORDINARY ENGLISH: EXAMPLES (REF-KECCN71).=
KIPSA69	COMPUTER AIDED	DATA	DIRECTORY DEFINITION (REF-TEDAR71).=SYSTEMS ANALYSIS DOCUMENTATION:
EVUDS71		DATA	STRUCTURES AND ALGORITHMIC DETERMINACY (REF-WIAFS71).=
GAPLV63	EW OF LANGUAGE	DATA	PROCESSING (REF-PAMFS71).=A LINGUISTICS VI
COJEM69	FOR AN ON-LINE	DATA	BASE UNDER TSS/360 (REF-KAAEU71).=ESTABLISHMENT AND MAINTENANCE OF A STORAGE H
DAOE65	MULTIPLE FILE	DATA	STORAGE SYSTEM : AN APPLICATION OF INTEGER PROGRAMMING (REF-WACOE71).=ON OPTIM
DOGED69	ELEMENTS OF	DATA	MANAGEMENT SYSTEMS (REF-GURER71).=
COKES69	ORITHM FOR THE	DATA	BASE OF A HUMAN BELIEF SYSTEM (REF-SUSMS71) (REF-PAMFS71) .=EXPERIMENTS WITH A
CODSG69	OF GENERALIZED	DATA	BASE MANAGEMENT SYSTEMS (REF-MKRS71) .=A SURVEY
KAADR67		DATA-RETRIEVAL	BY COMPUTER: A CRITICAL SURVEY (REF-SUSMS71) (REF-KUJQQ71).=
LOEDC68	E FOR CALENDAR	DATE	FOR MACHINE-TO-MACHINE DATA INTERCHANGE (REF-MUJDC71).=DATA COD
LOCMS66	STRUCTURES IN	DEACON	NATURAL LANGUAGE QUESTION-ANSWERING SYSTEM (REF-SUSMS71).=MEMORY
VEJED	CODES FOR THE	DECIMAL	NUMBER SYSTEM (REF-MUJDC71).=ERROR DETECTING AND CORRECTING
DIRRD71	A FILE AND THE	DECISION	PROBLEM FOR CLASSES OF PROPER FORMULAS.=THE RELATIONAL DAT
DIRRD69	ABILITY OF THE	DECISION	PROBLEM FOR THE CLASS OF DEFINITE FORMULAS (REF-DIRRD71).=THE RECURSIVE UNSOLV
ACWSC54	E CASES OF THE	DECISION	PROBLEM (REF-DIRRD71).=SOLVABL
GRMSG63	ERATION OUTPUT	DECOMPOSITION	METHOD (REF-TEDAR71).=SYSTEMS GEN
ASWTR68	PROCESSOR WITH	DEDUCTIVE	CAPABILITIES (REF-KECCN71).=TRAMP: AN INTERPRETIVE ASSOCIATIVE
HAGGG	RULES: A	DEFENSE	OF PHRASE STRUCTURE (REF-DOBHF71).=GENERATIVE GRAMMARS WITHOUT TRANSFORMATIONA
NACAD67	ACCURATELY	DEFINED	SYSTEMS (REF-TEDAR71).=
DIRRD69	R THE CLASS OF	DEFINITE	FORMULAS (REF-DIRRD71).=THE RECURSIVE UNSOLVABILITY OF THE DECISION PROBLEM FO
KIPSA69	DIRECTORY	DEFINITION	(REF-TEDAR71).=SYSTEMS ANALYSIS DOCUMENTATION: COMPUTER AIDED DATA
HAUDT64		DEPENDENCY	THEORY: A FORMALIZATION AND SOME OBSERVATIONS (REF-PAMFS71).=
SHSNS69	WERER:	DESCRIPTION	AND EXAMPLES (REF-SUSMS71).=A NET STRUCTURE BASED RELATIONAL QUESTION ANS
OLYTD68	INING A FORMAL	DESCRIPTION	OF ENGLISH (REF-PAMFS71).=TOWARD THE DEVELOPMENT OF COMPUTATIONAL AIDS FOR OBT
TUSD63	ATA PROCESSING	DESIGN	(REF-TEDAR71).=SEQUENTIAL D
TEUMD69	DOLOGY FOR THE	DESIGN	OF INFORMATION PROCESSING SYSTEMS (REF-TEDAR71).=METHO
TEURS69	VIEW OF SYSTEM	DESIGN	TECHNIQUES (REF-WACOE71).=A RE
WACOE71	EVALUATION OF	DESIGN	TRADEOFFS IN FILE SYSTEMS .=QUANTITATIVE
MAMRD66	L DATA FILE 1:	DESIGN	PHILOSOPHY (REF-DIRRD71).=RELATIONA
SEMFD69	FILE	DESIGN	HANDBOOK (REF-WACOE71).=
MAJDR67		DESIGN	OF REAL-TIME COMPUTER SYSTEMS (REF-TEDAR71).=
NUJDO61	ON THE	DESIGN	AND OPTIMIZATION OF INFORMATION PROCESSING SYSTEMS (REF-WACOE71).=
MCWTA69	THE AUTOMATIC	DESIGN	OF DATA ORGANIZATION FOR LARGE SCALE INFORMATION PROCESSING SYSTEMS (REF-WACOE
LEMDR68		DESIGN	OF A REVISED ON-LINE INFORMATION RETRIEVAL SYSTEM (REF-KAAEU71) .=
VEJED	ERROR	DETECTING	AND CORRECTING CODES FOR THE DECIMAL NUMBER SYSTEM (REF-MUJDC71).=
EVUDS71	ND ALGORITHMIC	DETERMINACY	(REF-WIAFS71).=DATA STRUCTURES A
WIAFS71	FILE STRUCTURE	DETERMINATION	=FILE S

MALMF69	FILE STRUCTURE	DETERMINATION	FOR LARGE ON-LINE DATA FILES (REF-WIAFS71).=A MODEL FOR
EALAU67	AUTOMATIC	DETERMINATION	OF PARTS OF SPEED OF ENGLISH WORDS (REF-PAMFS71).=
CLCFD66	FACTORS	DETERMINING	THE PERFORMANCE OF INDEXING SYSTEMS (REF-JADOC71).=
YOMIS69	CE: TOWARD THE	DEVELOPMENT	OF A TRUE SCIENTIFIC DISCIPLINE (REF-SUSMS71).=INFORMATION SCIEN
CLYTD68	TOWARD THE	DEVELOPMENT	OF COMPUTATIONAL AIDS FOR OBTAINING A FORMAL DESCRIPTION OF ENGLISH (REF-PAMFS
DOJ		DEVOTIONS	(REF-DOHHF71).=
KOMAQ69	S ABOUT SIMPLE	DIAGRAMS	(REF-SUSMS71).=AUTOMATIC QUESTION-ANSWERING OF ENGLISH-LIKE QUESTION
SCEDM63	A	DICTIONARY	FOR MINIMAL REDUNDANCY ENCODING (REF-MUJDC71).=
FERDS69		DICTIONARY	OF SYMBOLS OF MATHEMATICAL LOGIC (REF-KUJQQ71).=
BOAEN60	NEW METHOD OF	DICTIONARY	CONSTRUCTION (REF-KNGBT71).=ON THE EFFICIENCY OF A
KIGEL69	SCALE COMPUTER	DIRECT	ACCESS STORAGE MANIPULATION (REF-HUEHM71).=EXPERIMENTS IN LARGE
SUSDR69	A	DIRECTED	RANDOM PARAGRAPH GENERATOR (REF-SUSMS71).=
KIPSA69	D DATA	DIRECTORY	DEFINITION (REF-TEDAR71).=SYSTEMS ANALYSIS DOCUMENTATION: COMPUTER AIDE
YOMIS69	SCIENTIFIC	DISCIPLINE	(REF-SUSMS71).=INFORMATION SCIENCE: TOWARD THE DEVELOPMENT OF A TRUE
KOAOD66	AN OCCUPANCY	DISCIPLINE	AND APPLICATIONS, (REF-GURER71).=
TAMNU66	ON-UNIFORM KEY	DISTRIBUTION	AND ADDRESS CALCULATION SORTING (REF-GURER71).=N
CHYFT71	FULL TEXT	DOCUMENT	RETRIEVAL: HEBREW LEGAL TEXTS (REPORT ON THE FIRST PHASE OF THE RESPONSA RETRI
SAGSA65	MART AUTOMATIC	DOCUMENT	RETRIEVAL SYSTEM -- AN ILLUSTRATION (REF-JADOC71).=THE S
SGHSA64	FOR MACHANIZED	DOCUMENTATION	(REF-MKRS71) .=STATISTICAL ASSOCIATION METHODS
KIPSA69	STEMS ANALYSIS	DOCUMENTATION	COMPUTER AIDED DATA DIRECTORY DEFINITION (REF-TEDAR71).=SY
EISP68	MS PROGRAMMING	DOCUMENTATION	TABLES, REMOTE FILE MANAGEMENT SYSTEM, PHASE 1 (REF-WIAFS71).=SYSTE
MUJDC71	TECHNIQUES FOR	ECONOMIC	PROCESSING OF LARGE COMMERCIAL FILES.=DATA COMPRESSION
WHAPM50	MATICA (SECOND	EDITION)	(REF-KUJQQ71).=PRINCIPIA MATHE
SWJEI67		EFFECTIVENESS	OF INFORMATION RETRIEVAL METHODS (REF-JADOC71).=
BOAEN60	ON THE	EFFICIENCY	OF A NEW METHOD OF DICTIONARY CONSTRUCTION (REF-KNGBT71).=
KAAEU71		EFFICIENT	UTILIZATION OF LIMITED ACCESS ARCHIVAL STORAGE IN A TIME SHARED ENVIRONMENT.=
NEPEE61		EFFICIENT	ERROR-LIMITING VARIABLE-LENGTH CODES (REF-MUJDC71).=
KEAEI65		ELECTRONIC	INFORMATION HANDLING (REF-GURER71).=
GULEI63		ELECTRONIC	INFORMATION-LOGIC MACHINES (REF-GURER71).=
SCELE67	A LANGUAGE	ELEMENT	FOR COMPRESSION CODING (REF-MUJDC71).=
GURER71		ELEMENTS	OF THE RANDOMIZED COMBINATORIAL FILE STRUCTURE.=
REHES47		ELEMENTS	OF SYMBOLIC LOGIC (REF-KUJQQ71).=
DOGED69		ELEMENTS	OF DATA MANAGEMENT SYSTEMS (REF-GURER71).=
BEJIS63	RIEVAL: TOOLS,	ELEMENTS	THEORIES (REF-GURER71).=INFORMATION STORAGE AND RET
LOUIE68	NTRODUCTION TO	ELF	- LIST PROCESSING IN PL/I (REF-LODCC71).=AN I
WEJEC66		ELIZA	A COMPUTER PROGRAM FOR THE STUDY OF NATURAL LANGUAGE COMMUNICATION BETWEEN MAN
SCEDM63	MAL REDUNDANCY	ENCODING	(REF-MUJDC71).=A DICTIONARY FOR MINI
NIGSP69	ARCHING ATOMIC	ENERGY	LAW BY COMPUTER (REF-CHYFT71).=THE STATUS PROJECT:SE
WOWSI67	ERPRETATION OF	ENGLISH	QUESTIONS ON A STRUCTURED DATA BASE (REF-PAMFS71).=SEMANTIC INT
LOUCC71	ICTED, NATURAL	ENGLISH	=CUE: A CUSTOMIZED SYSTEM FOR RESTR
THUPS69	IC ANALYSIS OF	ENGLISH	SENTENCES (REF-PAMFS71).=A PROGRAM FOR THE SYNTACI
PRAIT69	ES IN MEDICAL	ENGLISH	(REF-PAMFS71).=IDENTIFICATION AND TRANSFORMATION OF TERMINAL MORPHEM
ROPIC66	ORE GRAMMAR OF	ENGLISH	(REF-LODCC71).=THE IHM C
MOUPR69	ON THE USE OF	ENGLISH	IN INFORMATION RETRIEVAL (REF-PAMFS71) .=A PROGRESS REPORT
SIKAE65	ANSWERING	ENGLISH	QUESTIONS BY COMPUTER: A SURVEY (REF-MKRS71) (REF-PAMFS71) (REF-SUSMS71) .=
STRIT68	AL THEORIES ON	ENGLISH	SYNTAX (REF-PAMFS71) .=INTEGRATION OF TRANSFORMATION
RONUE68	THE USE OF	ENGLISH	FOR COMMUNICATING WITH A COMPUTER (REF-LODCC71).=
PRAAP69	ING OF MEDICAL	ENGLISH	(REF-PAMFS71).=AUTOMATIC PROCESS
MITEP64	MITRE,	ENGLISH	PREPROCESSOR MANUAL (REF-LODCC71).=
CLYTD68	DESCRIPTION OF	ENGLISH	(REF-PAMFS71).=TOWARD THE DEVELOPMENT OF COMPUTATIONAL AIDS FOR OBTAINING A F
KECDM68	NT IN ORDINARY	ENGLISH	EXAMPLES (REF-KECCN71).=DATA MANAGEME
EALAU67	TS OF SPEED OF	ENGLISH	WORDS (REF-PAMFS71).=AUTOMATIC DETERMINATION OF PAR
FRCSE52	E STRUCTURE OF	ENGLISH	(REF-DOHHF71).=TH
DOGRE69	GE SYSTEM: REL	ENGLISH	(REF-DOHHF71).=A RAPIDLY EXTENSIBLE LANGUA
CHYAG67	HEBREW)	ENGLISH	ABSTRACT (REF-CHYFT71).=AUTOMATIC GRAMMATICAL ANALYSIS OF THE HEBREW VERB (IN
CUPAR69	ON GRAMMAR FOR	ENGLISH	(REF-LODCC71).=AN AUTOMATED RECOGNTI
KOMAQ69	N-ANSWERING OF	ENGLISH-LIKE	QUESTIONS ABOUT SIMPLE DIAGRAMS (REF-SUSMS71).=AUTOMATIC QUESTIO

CHWTR62	A 'SEMIFORMAL'	ENGLISH-LIKE	LANGUAGE. (REF-MKRS71).=TRANSLATION OF RETRIEVAL REQUESTS COUCHED IN
KAAEU71	A TIME SHARED	ENVIRONMENT	=EFFICIENT UTILIZATION OF LIMITED ACCESS ARCHIVAL STORAGE IN
VEJED		ERROR	DETECTING AND CORRECTING CODES FOR THE DECIMAL NUMBER SYSTEM (REF-MUJDC71).=
KEWET68		ERRORS	IN TRANSMISSION OF COMPRESSED DATA (REF-MUJDC71).=
NEPEE61	EFFICIENT	ERROR-LIMITING	VARIABLE-LENGTH CODES (REF-MUJDC71).=
COJEM69		ESTABLISHMENT	AND MAINTENANCE OF A STORAGE HIERARCHY FOR AN ON-LINE DATA BASE UNDER TSS/360
WACQE71	QUANTITATIVE	EVALUATION	OF DESIGN TRADEOFFS IN FILE SYSTEMS .=
SEMF068	E ORGANIZATION	EVALUATION	MODEL-FOREM (REF-WACQF71).=A FIL
SAGCE68	COMPUTER	EVALUATION	FOR INDEXING AND TEXT PROCESSING (REF-JADOC71).=
SAGRA69	TRIEVAL SYSTEM	EVALUATION	(REF-JADOC71).=RELEVANCE ASSESSMENTS AND RE
SAGEP70		EVALUATION	PROBLEMS IN INTERACTIVE INFORMATION RETRIEVAL (REF-SUSMS71).=
SHSNS69	ESCRIPTION AND	EXAMPLES	(REF-SUSMS71).=A NET STRUCTURE BASED RELATIONAL QUESTION ANSWERER: D
KECDM68	INARY ENGLISH:	EXAMPLES	(REF-KECCN71).=DATA MANAGEMENT IN ORD
LERRD69	AND INFERENCE	EXEC	(REF-DIRRD71).=RELATIONAL DATA FILE: EXPERIENCE WITH A SYSTEM FOR PROPOSITION
LERCS67	FOR INFERENCE	EXECUTION	AND DATA RETRIEVAL (REF-DIRRD71).=A COMPUTER SYSTEM
LERRD65	NFERENCE	EXECUTION	AND DATA RETRIEVAL (REF-KUJQQ71).=RELATIONAL DATA FILE: A TOOL FOR MECHANIZED
LERCS67	FOR INFERENCE	EXECUTION	AND DATA RETRIEVAL (REF-KUJQQ71).=A COMPUTER SYSTEM
LERTM65	NFERENCE	EXECUTION	AND DATA RETRIEVAL (REF-KECCN71).=RELATIONAL DATA FILE: A TOOL FOR MECHANIZED
LERRD69	NAL DATA FILE:	EXPERIENCE	WITH A SYSTEM FOR PROPOSITIONAL DATA STORAGE AND INFERENCE EXEC (REF-DIRRD71).
OVCC165	ATION TRANSFER	EXPERIMENTS	(INTREX) (REF-MKRS71) .=PLANNING CONFERENCE ON INFORM
K1GEL69		EXPERIMENTS	IN LARGE SCALE COMPUTER DIRECT ACCESS STORAGE MANIPULATION (REF-HUEHM71).=
KAMEP67		EXPERIMENTS	WITH A POWERFUL PARSER (REF-DOBHF71) (REF-KECCN71).=
COKES69		EXPERIMENTS	WITH A SEARCH ALGORITHM FOR THE DATA BASE OF A HUMAN BELIEF SYSTEM (REF-SUSMS7
QUWVE60	VARIABLES	EXPLAINED	AWAY (REF-KECCN71).=
WEUES66		EXPLORATIONS	IN SEMANTIC THEORY (REF-PAMFS71).=
THFRE69	REL: A RAPIDLY	EXTENSIBLE	LANGUAGE SYSTEM (REF-DOBHF71).=REL: A
LOPRE69	A RAPIDLY	EXTENSIBLE	LANGUAGE SYSTEM: THE REL LANGUAGE PROCESSOR (REF-DOBHF71).=
LOPRE69	A RAPIDLY	EXTENSIBLE	LANGUAGE SYSTEM (REF-SUSMS71).=
DOGRE69	A RAPIDLY	EXTENSIBLE	LANGUAGE SYSTEM: REL ENGLISH (REF-DOBHF71).=
DACE65	ON OPTIMAL	EXTRACING	FROM A MULTIPLE FILE DATA STORAGE SYSTEM : AN APPLICATION OF INTEGER PROGRAMMI
ABRIM69	: A METHOD FOR	EXTRACTING	MEANING FROM INPUT SENTENCES (REF-PAMFS71).=IMPLICATIONAL MOLECULES
ABRIM69	: A METHOD FOR	EXTRACTING	MEANING FROM INPUT SENTENCES (REF-SUSMS71).=IMPLICATIONAL MOLECULES
IBMDM70	ATA MANAGEMENT	FACILITIES	OF TSS/360 (REF-KAAEU71).=D
BRWGP66	AL INFORMATION	FACILITY	(REF-SUSMS71).=GUIDE TO THE PROCESSING, STORAGE AND RETRIEVAL OF BIBLIOGRAPHI
CLCFD66		FACTORS	DETERMINING THE PERFORMANCE OF INDEXING SYSTEMS (REF-JADOC71).=
DOBHF71	HOW	FEATURES	RESOLVE SYNTACTIC AMBIGUITY.=
DIRRD71	ELATIONAL DATA	FILE	AND THE DECISION PROBLEM FOR CLASSES OF PROPER FORMULAS.=THE R
WIAFS71		FILE	STRUCTURE DETERMINATION.=
GURER71	COMBINATORIAL	FILE	STRUCTURE.=ELEMENTS OF THE RANDOMIZED
TEDAR71	TO RESEARCH IN	FILE	ORGANIZATION.=AN APPROACH
WIAMC70	COMPARISON OF	FILE	ORGANIZATION AND PROCESSING PROCEDURES FOR HIERARCHICAL STORAGE STRUCTURES (RE
WACQE71	N TRADEOFFS IN	FILE	SYSTEMS .=QUANTITATIVE EVALUATION OF DESIG
MAMRD66	ELATIONAL DATA	FILE	I: DESIGN PHILOSOPHY (REF-DIRRD71).=R
SEMF068	A	FILE	ORGANIZATION EVALUATION MODEL-FOREM (REF-WACQE71).=
MAWFC68		FILE	COMPRESSION USING HUFFMAN CODING (REF-MUJDC71).=
SEMFD69		FILE	DESIGN HANDBOOK (REF-WACQE71).=
LERRD66	ELATIONAL DATA	FILE	II: IMPLEMENTATION (REF-DIRRD71).=R
LERRD69	ELATIONAL DATA	FILE	EXPERIENCE WITH A SYSTEM FOR PROPOSITIONAL DATA STORAGE AND INFERENCE EXEC (R
LEDFS69		FILE	STRUCTURES FOR ON-LINE SYSTEMS (REF-HUEHM71).=
KUJIR70	ELATIONAL DATA	FILE	REMARKS ON THE ADMISSIBILITY OF INPUT QUERIES (REF-DIRRD71).=INTERROGATING A
HAMAC63	NG THEORY TO A	FILE	ADDRESS PROBLEM (REF-GURER71).=AN APPLICATION OF CODI
MALMF69	A MODEL FOR	FILE	STRUCTURE DETERMINATION FOR LARGE ON-LINE DATA FILES (REF-WIAFS71).=
LERTM65	ELATIONAL DATA	FILE	A TOOL FOR MECHANIZED INFERENCE EXECUTION AND DATA RETRIEVAL (REF-KECCN71).=R
LEDFS67		FILE	STRUCTURES FOR ON-LINE SYSTEMS (REF-GURER71).=
KAMMS70	F THE SEMANTIC	FILE	(REF-SUSMS71).=THE MIND SYSTEM: THE STRUCTURE O
LERRD65	ELATIONAL DATA	FILE	A TOOL FOR MECHANIZED INFERENCE EXECUTION AND DATA RETRIEVAL (REF-KUJQQ71).=R

LEDFS69		FILE	STRUCTURES FOR ON-LINE SYSTEMS (REF-WIAFS71).=
KUJIR70	ELATIONAL DATA	FILE	REMARKS ON THE ADMISSIBILITY OF INPUT QUERIES (REF-KUJQQ71).=INTERROGATING A
DAGI68	MATION, REMOTE	FILE	MANAGEMENT SYSTEM, PHASE I (REF-WIAFS71).=GENERAL INFOR
GHSF069		FILE	ORGANIZATION: ON THE SELECTION OF RANDOM ACCESS INDEX POINTS FOR SEQUENTIAL FI
EISP68	TABLES, REMOTE	FILE	MANAGEMENT SYSTEM, PHASE 1 (REF-WIAFS71).=SYSTEMS PROGRAMMING DOCUMENTATION,
DAOE65	ROM A MULTIPLE	FILE	DATA STORAGE SYSTEM : AN APPLICATION OF INTEGER PROGRAMMING (REF-WACOE71).=ON
FIJGF70	A GENERALIZED	FILE	MANAGEMENT SYSTEM (REF-HUEHM71).=
CAJUP65	TSBURGH LINEAR	FILE	TANDEM SYSTEM (REF-GURER71).=UPLIFTS-UNIVERSITY OF PIT
GHSAF68	TE GEOMETRY IN	FILE	ORGANIZATION FOR RECORDS WITH MULTIPLE-VALUED ATTRIBUTES (REF-GURER71).=APPLIC
GSRRC69	COMBINATORIAL	FILE	STRUCTURE FOR STORAGE AND RETRIEVAL SYSTEMS (REF-GURER71).=A RANDOMIZED
AFCFF67	FORMATTED	FILE	ORGANIZATION TECHNIQUES (REF-WACOE71).=
CHNCF69	COMPARISON OF	FILE	ORGANIZATION TECHNIQUES (REF-WIAFS71).=A
BUWF063		FILE	ORGANIZATION AND ADDRESSING (REF-WACOE71).=
MUJDC71	RGE COMMERCIAL	FILES	=DATA COMPRESSION TECHNIQUES FOR ECONOMIC PROCESSING OF LA
MALMF69	ON-LINE DATA	FILES	(REF-WIAFS71).=A MODEL FOR FILE STRUCTURE DETERMINATION FOR LARGE
HSDFS70	RETRIEVAL FROM	FILES	(REF-GURER71).=A FORMAL SYSTEM FOR INFORMATION
GHSF069	FOR SEQUENTIAL	FILES	(REF-WACOE71) (REF-KNGBT71).=FILE ORGANIZATION: ON THE SELECTION OF RANDOM AC
DZBOS65	OF STRUCTURED	FILES	(REF-GURER71).=THE ORGANIZATION
GHSAF68	APPLICATION OF	FINITE	GEOMETRY IN FILE ORGANIZATION FOR RECORDS WITH MULTIPLE-VALUED ATTRIBUTES (REF
CHYFT71	(REPORT ON THE	FIRST	PHASE OF THE RESPONSA RETRIEVAL PROJECT).=FULL TEXT DOCUMENT RETRIEVAL: HEBREW
WIPIF60	TREES,	FORESTS	AND REARRANGING (REF-KNGBT71).=
CHBSH63	: A REALIZABLE	FORM	OF ASSOCIATIVE MEMORY (REF-GURER71).=SHIEF
OLYTD68	OR OBTAINING A	FORMAL	DESCRIPTION OF ENGLISH (REF-PAMFS71).=TOWARD THE DEVELOPMENT OF COMPUTATIONAL
HSDFS70	A	FORMAL	SYSTEM FOR INFORMATION RETRIEVAL FROM FILES (REF-GURER71).=
GRCSI70	TERROGATION OF	FORMAL	CONTROL SYSTEMS (REF-TEDAR71).=SPECIFICATION AND IN
SCRTC69	COMPUTATIONAL	FORMALIZATION	OF NATURAL LANGUAGE SEMANTICS (REF-PAMFS71).=TOWARDS A
HADDT64	ENCY THEORY: A	FORMALIZATION	AND SOME OBSERVATIONS (REF-PAMFS71).=DEPEND
AFCFF67		FORMATTED	FILE ORGANIZATION TECHNIQUES (REF-WACOE71).=
DIRRD71	SSES OF PROPER	FORMULAS	=THE RELATIONAL DATA FILE AND THE DECISION PROBLEM FOR CLA
DIRRU69	SS OF DEFINITE	FORMULAS	(REF-DIRRD71).=THE RECURSIVE UNSOLVABILITY OF THE DECISION PROBLEM FOR THE CL
YOJAF59	ABSTRACT	FORMULATION	OF DATA PROCESSING PROBLEMS (REF-TEDAR71).=
FEAF158		FOUNDATIONS	OF INFORMATION THEORY (REF-KAAEU71).=
CHTGI68	MS FOR CONTEXT	FREE	GRAMMAR (REF-LODCC71).=ON THE GENERATION AND IMPLEMENTATION OF REDUCTIONS ANAL
PAMFS71	THE	FUNCTION	OF SEMANTICS IN AUTOMATED LANGUAGE PROCESSING.=
JADNS69	TE ON A SET OF	FUNCTIONS	FOR INFORMATION RETRIEVAL (REF-JADOC71).=A NO
KNDAC68	AMMING, VOL 1:	FUNDAMENTAL	ALGORITHMS (REF-KNGBT71).=THE ART OF COMPUTER PROGR
DAGI68		GENERAL	INFORMATION, REMOTE FILE MANAGEMENT SYSTEM, PHASE I (REF-WIAFS71).=
MKRGD69		GENERALIZED	DATA MANAGEMENT SYSTEMS - SOME PRESPECTIVES (REF-MKRS71).=
FIJGF70	A	GENERALIZED	FILE MANAGEMENT SYSTEM (REF-HUEHM71).=
COUSG69	A SURVEY OF	GENERALIZED	DATA BASE MANAGEMENT SYSTEMS (REF-MKRS71) .=
EAJGR65		GENERATING	A RECOGNIZER FOR A BNF GRAMMAR (REF-DOBHF71).=
GRMSG63	SYSTEMS	GENERATION	OUTPUT DECOMPOSITION METHOD (REF-TEDAR71).=
CHTGI68	ON THE	GENERATION	AND IMPLEMENTATION OF REDUCTIONS ANALYSIS PROGRAMS FOR CONTEXT FREE GRAMMAR (R
HAGGG		GENERATIVE	GRAMMARS WITHOUT TRANSFORMATIONAL RULES: A DEFENSE OF PHRASE STRUCTURE (REF-DO
SUSDR69	NDOM PARAGRAPH	GENERATOR	(REF-SUSMS71).=A DIRECTED RA
KAJLG68	A LINEAR	GEOGRAPHICAL	CODE FOR MANAGEMENT INFORMATION SYSTEMS (REF-MUJDC71).=
EVTHP64	OGRAM TO SOLVE	GEOMETRIC-ANALOGY	PROBLEMS (REF-JADOC71).=A HEURISTIC PR
GHSAF68	TION OF FINITE	GEOMETRY	IN FILE ORGANIZATION FOR RECORDS WITH MULTIPLE-VALUED ATTRIBUTES (REF-GURER71)
ROPIC66	THE IBM CORE	GRAMMAR	OF ENGLISH (REF-LODCC71).=
JEOPG65	PHILOSOPHY OF	GRAMMAR	(REF-KUJQQ71).=THE
LASOS66	RATIFICATIONAL	GRAMMAR	(REF-PAMFS71).=AN OUTLINE OF ST
EAJGR65	IZER FOR A BNF	GRAMMAR	(REF-DOBHF71).=GENERATING A RECOGN
CHTGI68	R CONTEXT FREE	GRAMMAR	(REF-LODCC71).=ON THE GENERATION AND IMPLEMENTATION OF REDUCTIONS ANALYSIS PR
CUPAR69	TEU RECOGNTION	GRAMMAR	FOR ENGLISH (REF-LODCC71).=AN AUTOMA
WOWTN70	SITION NETWORK	GRAMMARS	FOR NATURAL LANGUAGE ANALYSIS (REF-KECCN71) (REF-DOBHF71) .=TRAN
HAGGG	GENERATIVE	GRAMMARS	WITHOUT TRANSFORMATIONAL RULES: A DEFENSE OF PHRASE STRUCTURE (REF-DOBHF71).=
KULCG63	CODING OF	GRAMMATICAL	DATA (REF-MUJDC71).=

CHYAG67	AUTOMATIC	GRAMMATICAL	ANALYSIS OF THE HEBREW VERB (IN HEBREW) ENGLISH ABSTRACT (REF-CHYFT71).=
CHYAG66	AUTOMATIC	GRAMMATICAL	ANALYSIS OF THE HEBREW VERB (IN HEBREW) (REF-CHYFT71).=
STISK63	A MAN-MACHINE	GRAPHICAL	COMMUNICATION SYSTEM (REF-HUEHM71) .=SKETCHPAD,
IBMTA68	TIME AUTOMATED	GRID	SYSTEM (TAG): SALES AND SYSTEMS GUIDE (REF-TEDAR71).=THE
IBMTA68	ND SYSTEMS	GUIDE	(REF-TEDAR71).=THE TIME AUTOMATED GRID SYSTEM (TAG): SALES A
FIJUG70	A USERS	GUIDE	TO THE OPERATION OF AN INFORMATION STORAGE AND RETRIEVAL SYSTEM ON THE B5500 C
BRWGP66		GUIDE	TO THE PROCESSING, STORAGE AND RETRIEVAL OF BIBLIOGRAPHIC INFORMATION AT THE N
SEMFD69	FILE DESIGN	HANDBOOK	(REF-WACQE71).=
KEAEI65	IC INFORMATION	HANDLING	(REF-GURER71).=ELECTRON
BOCMI63	OF INFORMATION	HANDLING	(REF-GURER71).=METHODS
HUEHM71	A	HEATHKIT	METHOD FOR BUILDING DATA MANAGEMENT PROGRAMS.=
CHYFT71	ENT RETRIEVAL:	HEBREW	LEGAL TEXTS (REPORT ON THE FIRST PHASE OF THE RESPONSA RETRIEVAL PROJECT).=FUL
SCACS69	INDEXING: THE	HEBREW	RESPONSA PROJECT (REF-CHYFT71).=COMPUTERIZED STORAGE AND RETRIEVAL OF CASE LAW
FRACI69	LIARY TOOLS IN	HEBREW	LEGAL RESEARCH (IN HEBREW) (REF-CHYFT71).=THE COMPUTER AND INFORMATION RETRI
FRAC169	RESEARCH (IN	HEBREW	) (REF-CHYFT71).=THE COMPUTER AND INFORMATION RETRIEVAL AS AUXILIARY TOOLS IN
CHYAG67	NALYSIS OF THE	HEBREW	VERB (IN HEBREW) ENGLISH ABSTRACT (REF-CHYFT71).=AUTOMATIC GRAMMATICAL A
CHYAG66	NALYSIS OF THE	HEBREW	VERB (IN HEBREW) (REF-CHYFT71).=AUTOMATIC GRAMMATICAL A
COMMS70	RETRIEVAL (IN	HEBREW	) (REF-CHYFT71).=FULL TEXT INFORMATION
SLMMS69	H RESPONSA (IN	HEBREW)	(REF-CHYFT71).=MARRANOS SEARCH IN THE RIVAS
SLMOR70	H RESPONSA (IN	HEBREW)	(REF-CHYFT71).=THE OCCULT IN THE RIVAS
CHYAG67	VERB (IN	HEBREW)	ENGLISH ABSTRACT (REF-CHYFT71).=AUTOMATIC GRAMMATICAL ANALYSIS OF THE HEBREW
CHYAG66	VERB (IN	HEBREW)	(REF-CHYFT71).=AUTOMATIC GRAMMATICAL ANALYSIS OF THE HEBREW
EVTHP64	A	HEURISTIC	PROGRAM TO SOLVE GEOMETRIC-ANALOGY PROBLEMS (REF-JADOC71).=
WIAMC70	PROCEDURES FOR	HIERARCHICAL	STORAGE STRUCTURES (REF-WIAFS71).=A METHODOLOGY FOR COMPARISON OF FILE ORGANIZ
GLEMP70	ORE PARSER FOR	HIERARCHICAL	SYNTACTIC STRUCTURES (REF-PAMFS71).=THE MULTIST
COJEM69	E OF A STORAGE	HIERARCHY	FOR AN ON-LINE DATA BASE UNDER TSS/360 (REF-KAAEU71).=ESTABLISHMENT AND MAINTE
BOUQA64	ING SYSTEM FOR	HIGH	SCHOOL ALGEBRA WORK PROBLEMS (REF-KUJQQ71) (REF-SUSMS71) (REF-LOUCC71).=A QUES
PAMHC63		HOMOGRAPHS	I: CLASSIFICATION AND IDENTIFICATION (REF-PAMFS71).=
PAMHC67		HOMOGRAPHS	II: CLASSIFICATION AND IDENTIFICATION (REF-PAMFS71).=
MAWFC68	PRESSION USING	HUFFMAN	CODING (REF-MUJDC71).=FILE COM
REACH64	CONSISTENCY OF	HUMAN	JUDGEMENTS OF RELEVANCE (REF-JADOC71).=THE
COKES69	ATA BASE OF A	HUMAN	BELIEF SYSTEM (REF-SUSMS71) (REF-PAMFS71) .=EXPERIMENTS WITH A SEARCH ALGORITH
ROPIC66	THE	IBM	CORE GRAMMAR OF ENGLISH (REF-LODCC71).=
LUHKW59	ORD-IN-CONTEXT	IDEA	FOR TECHNICAL LITERATURE (KWIC INDEX). (REF-MKRS71).=KEY-W
PAMHC63	SIFICATION AND	IDENTIFICATION	(REF-PAMFS71).=HOMOGRAPHS I: CLAS
PAMHC67	SIFICATION AND	IDENTIFICATION	(REF-PAMFS71).=HOMOGRAPHS II: CLAS
PRAIT69		IDENTIFICATION	AND TRANSFORMATION OF TERMINAL MORPHEMES IN MEDICAL ENGLISH (REF-PAMFS71).=
KECPA70	IONAL STANDARD	IDENTIFICATION	OF STATES.....FOR INFORMATION INTERCHANGE (REF-MUJDC71).=PROPOSED AMERICAN N
GRCPI69	NCIPLES OF THE	IDENTIFICATION	OF INFORMATION (REF-TEDAR71).=PRI
SAGSA65	-- AN	ILLUSTRATION	(REF-JADOC71).=THE SMART AUTOMATIC DOCUMENT RETRIEVAL SYSTEM
KIGAL70	NGUAGE AND ITS	IMPLEMENTATION	(REF-HUEHM71).=APL/B5500: THE LA
LERRD66	DATA FILE II:	IMPLEMENTATION	(REF-DIRRD71).=RELATIONAL
CHTGI68	GENERATION AND	IMPLEMENTATION	OF REDUCTIONS ANALYSIS PROGRAMS FOR CONTEXT FREE GRAMMAR (REF-LOUCC71).=ON THE
BOUST68	RY IN COMPUTER	IMPLEMENTATIONS	(REF-PAMFS71).=SYNTACTIC THEO
ABRIM69		IMPLICATIONAL	MOLECULES: A METHOD FOR EXTRACTING MEANING FROM INPUT SENTENCES (REF-PAMFS71).
ABRIM69		IMPLICATIONAL	MOLECULES: A METHOD FOR EXTRACTING MEANING FROM INPUT SENTENCES (REF-SUSMS71).
JADBI69	BASIS FOR AN	IMPROVABILITY	MEASURE IN INFORMATION RETRIEVAL (REF-JADOC71).=

WRMML60	NIZING A LARGE	INDEX	APPENDIX: THE SOUNDEX CODE (REF-MUJDC71).=MECHA
GHSFO69	RANDOM ACCESS	INDEX	POINTS FOR SEQUENTIAL FILES (REF-WACQE71) (REF-KNGBT71).=FILE ORGANIZATION: O
BEJOA69	AN AID TO THE	INDEXER/CATALOGER	(REF-CHYFT71).=ON-LINE ACCESS TO INFORMATION: USE AS
LUVMR70	WITH COMBINED	INDEXES	(REF-GURER71).=MULTI-ATTRIBUTE RETRIEVAL
SAGCE68	EVALUATION FOR	INDEXING	AND TEXT PROCESSING (REF-JADOC71).=COMPUTER
SCACS69	W WITHOUT	INDEXING	THE HEBREW RESPONSA PROJECT (REF-CHYFT71).=COMPUTERIZED STORAGE AND RETRIEVAL
STMAI70	AUTOMATIC	INDEXING	A STATE OF THE ART REPORT (REF-MKRS71).=
DOLIA62		INDEXING	AND ABSTRACTING BY ASSOCIATION (REF-JADOC71).=
CLCFD66	PERFORMANCE OF	INDEXING	SYSTEMS (REF-JADOC71).=FACTORS DETERMINING THE
LUHKW59	TERATURE (KWIC	INDEX)	(REF-MKRS71).=KEY-WORD-IN-CONTEXT IDEA FOR TECHNICAL LI
LERRD69	TA STORAGE AND	INFERENCE	EXEC (REF-DIRRD71).=RELATIONAL DATA FILE: EXPERIENCE WITH A SYSTEM FOR PROPOSI
LERRD65	FOR MECHANIZED	INFERENCE	EXECUTION AND DATA RETRIEVAL (REF-KUJQQ71).=RELATIONAL DATA FILE: A TOOL
LERCS67	TER SYSTEM FOR	INFERENCE	EXECUTION AND DATA RETRIEVAL (REF-KUJQQ71).=A COMPU
LERCS67	TER SYSTEM FOR	INFERENCE	EXECUTION AND DATA RETRIEVAL (REF-DIRRD71).=A COMPU
LERTM65	FOR MECHANIZED	INFERENCE	EXECUTION AND DATA RETRIEVAL (REF-KECCN71).=RELATIONAL DATA FILE: A TOOL
MKRS71	R THE 1971 ACM	INFORMATION	STORAGE AND RETRIEVAL SYMPOSIUM .=INTRODUCTION AND PERSPECTIVES FO
YOMIS69		INFORMATION	SCIENCE: TOWARD THE DEVELOPMENT OF A TRUE SCIENTIFIC DISCIPLINE (REF-SUSMS71).
WACIR65		INFORMATION	RETRIEVAL (REF-GURER71).=
MECAI67	HE ANALYSIS OF	INFORMATION	SYSTEMS (REF-GURER71).=T
SEMIS69		INFORMATION	STORAGE AND RETRIEVAL SYSTEMS (REF-CHYFT71).=
SCG67		INFORMATION	RETRIEVAL: A CRITICAL VIEW (REF-GURER71).=
SAGDM66	AUTOMATIC	INFORMATION	RETRIEVAL (REF-GURER71).=DATA MANIPULATION AND PROGRAMMING PROBLEMS IN
SAGEP70	IN INTERACTIVE	INFORMATION	RETRIEVAL (REF-SUSMS71).=EVALUATION PROBLEMS
SWJEI67	FECTIVENESS OF	INFORMATION	RETRIEVAL METHODS (REF-JADOC71).=EF
OVCCI65	CONFERENCE ON	INFORMATION	TRANSFER EXPERIMENTS (INTREX) (REF-MKRS71) .=PLANNING
TEUMD69	THE DESIGN OF	INFORMATION	PROCESSING SYSTEMS (REF-TEDAR71).=METHODOLOGY FOR
MIMSI69	SEMANTIC	INFORMATION	PROCESSING (REF-SUSMS71).=
SPJIS65		INFORMATION	SYSTEMS SCIENCES: PROCEEDING OF THE SECOND CONGRESS (REF-GURER71).=
NUJDO61	PTIMIZATION OF	INFORMATION	PROCESSING SYSTEMS (REF-WACQE71).=ON THE DESIGN AND O
SAGAI63	AUTOMATIC	INFORMATION	ORGANIZATION AND RETRIEVAL (REF-MKRS71) (REF-GURER71).=
NUWCW68	TECHNIQUES FOR	INFORMATION	RETRIEVAL (REF-MUJDC71).=COMPRESSION WORD CODING
KEAEI65	ELECTRONIC	INFORMATION	HANDLING (REF-GURER71).=
LEMDR68	EVISED ON-LINE	INFORMATION	RETRIEVAL SYSTEM (REF-KAAEU71) .=DESIGN OF A R
JADNS69	FUNCTIONS FOR	INFORMATION	RETRIEVAL (REF-JADOC71).=A NOTE ON A SET OF
JADBI69	ITY MEASURE IN	INFORMATION	RETRIEVAL (REF-JADOC71).=BASIS FOR AN IMPROVABIL
MOJNL69		INFORMATION	RETRIEVAL AND NATURAL LANGUAGE (REF-PAMFS71) .=
MOJPR69	OF ENGLISH IN	INFORMATION	RETRIEVAL (REF-PAMFS71) .=A PROGRESS REPORT ON THE USE
JADCR	RELEVANCE AND	INFORMATION	RETRIEVAL (REF-JADOC71).=CLASSIFICATION,
KECPA70	ATES.....FOR	INFORMATION	INTERCHANGE (REF-MUJDC71).=PROPOSED AMERICAN NATIONAL STANDARD IDENTIFICATION
MCWTA69	R LARGE SCALE	INFORMATION	PROCESSING SYSTEMS (REF-WACQE71).=TOWARD THE AUTOMATIC DESIGN OF DATA ORGANIZA
KAJLG68	FOR MANAGEMENT	INFORMATION	SYSTEMS (REF-MUJDC71).=A LINEAR GEOGRAPHICAL CODE
KEATM62	ON MECHANIZED	INFORMATION	RETRIEVAL (REF-GURER71).=TEXTBOOK
LASA63	THE THEORY OF	INFORMATION	SYSTEMS (REF-WACQE71).=SOME APPROACHES TO
LABTA66	AL ANALYSIS OF	INFORMATION	SYSTEMS (REF-TEDAR71).=THEORETIC
GRCPI69	NTIFICATION OF	INFORMATION	(REF-TEDAR71).=PRINCIPLES OF THE IDE
CORIS69		INFORMATION	STORAGE AND RETRIEVAL (REF-SUSMS71).=

FEAFI58 FOUNDATIONS OF INFORMATION THEORY (REF-KAAEU71).=FOUNDA
FRACI69 E COMPUTER AND INFORMATION RETRIEVAL AS AUXILIARY TOOLS IN HEBREW LEGAL RESEARCH (IN HEBREW) (REF-CHYFT
FARTI61 RANSMISSION OF INFORMATION (REF-KAAEU71).=T
FRALI68 LEGAL INFORMATION RETRIEVAL (REF-CHYFT71).=
HSUFS70 MAL SYSTEM FOR INFORMATION RETRIEVAL FROM FILES (REF-GURER71).=A FOR
FOCIS65 INFORMATION STORAGE AND RETRIEVAL USING AVL TREES (REF-KNGBT71).=
FIJUG70 PERATION OF AN INFORMATION STORAGE AND RETRIEVAL SYSTEM ON THE B5500 COMPUTER (REF-HUEHM71).=A USERS GUID
DAGI68 GENERAL INFORMATION REMOTE FILE MANAGEMENT SYSTEM, PHASE I (REF-WIAFS71).=
BRWGP66 TECHNICAL INFORMATION FACILITY (REF-SUSMS71).=GUIDE TO THE PROCESSING, STORAGE AND RETRIEVAL OF BIBL
BOCMI63 METHODS OF INFORMATION HANDLING (REF-GURER71).=
BRWGP66 BIBLIOGRAPHIC INFORMATION AT THE NASA SCIENTIFIC AND TECHNICAL INFORMATION FACILITY (REF-SUSMS71).=GUIDE
COMMS70 FULL TEXT INFORMATION RETRIEVAL (IN HEBREW) (REF-CHYFT71).=
CHCIT61 INFORMATION THEORY (REF-GURER71).=
BEJIS63 INFORMATION STORAGE AND RETRIEVAL: TOOLS, ELEMENTS, THEORIES (REF-GURER71).=
COUIA62 AN INFORMATION ALGEBRA PHASE I REPORT (REF-TEDAR71).=
BEJOA69 LINE ACCESS TO INFORMATION NSF AS AN AID TO THE INDEXER/CATALOGER (REF-CHYFT71).=ON-
ARJIS69 INFORMATION SYSTEMS IN PERSPECTIVE (REF-GURER71).=
ASKIT65 INFORMATION THEORY (REF-KAAEU71).=
CGMIE58 INFORMATION CONTROL (REF-MKRS71).=
ADGAO RGANIZATION OF INFORMATION (REF-KNGBT71).=AN ALGORITHM FOR THE O
GULEI63 ELECTRONIC INFORMATION-LOGIC MACHINES (REF-GURER71).=
KUJIR70 MISSIBILITY OF INPUT QUERIES (REF-KUJQQ71).=INTERROGATING A RELATIONAL DATA FILE: REMARKS ON THE AD
KUJIR70 MISSIBILITY OF INPUT QUERIES (REF-DIRRD71).=INTERROGATING A RELATIONAL DATA FILE: REMARKS ON THE AD
COLOL68 AND PICTORIAL INPUT (REF-SUSMS71).=AN ON-LINE QUESTION-ANSWERING SYSTEM WITH NATURAL LANGUAGE
ABRIM69 EANING FROM INPUT SENTENCES (REF-PAMFS71).=IMPLICATIONAL MOLECULES: A METHOD FOR EXTRACTING M
ABRIM69 EANING FROM INPUT SENTENCES (REF-SUSMS71).=IMPLICATIONAL MOLECULES: A METHOD FOR EXTRACTING M
RUBIM62 AN INQUIRY INTO MEANING AND TRUTH (REF-KUJQQ71).=
DAOE65 APPLICATION OF INTEGER PROGRAMMING (REF-WACQE71).=ON OPTIMAL EXTRACING FROM A MULTIPLE FILE DATA STOR
STRIT68 INTEGRATION OF TRANSFORMATIONAL THEORIES OF SYNTAX (REF-KECCN71).=
STRIT68 INTEGRATION OF TRANSFORMATIONAL THEORIES ON ENGLISH SYNTAX (REF-PAMFS71) .=
SUSST69 ORY BASED UPON INTERACTIVE MEANING (REF-SUSMS71).=A SEMANTIC THE
SAGEP70 ON PROBLEMS IN INTERACTIVE INFORMATION RETRIEVAL (REF-SUSMS71).=EVALUATI
KECPA70 OR INFORMATION INTERCHANGE (REF-MUJDC71).=PROPOSED AMERICAN NATIONAL STANDARD IDENTIFICATION OF STATES..
LOEDC68 CHINE DATA INTERCHANGE (REF-MUJDC71).=DATA CODE FOR CALENDAR DATE FOR MACHINE-TO-MA
OTMIL69 INTERMEDIATE LANGUAGES FOR AUTOMATIC LANGUAGES PROCESSING (REF-PAMFS71?.=
FLIAI61 ANALYSIS OF INTERNAL COMPUTER SORTING (REF-WIAFS71).=
WOWS167 SEMANTIC INTERPRETATION OF ENGLISH QUESTIONS ON A STRUCTURED DATA BASE (REF-PAMFS71).=
ASWTR68 TRAMP: AN INTERPRETIVE ASSOCIATIVE PROCESSOR WITH DEDUCTIVE CAPABILITIES (REF-KECCN71).=
KUJIR70 INTERROGATING A RELATIONAL DATA FILE: REMARKS ON THE ADMISSIBILITY OF INPUT QUERIES (REF-KUJ
KUJIR70 INTERROGATING A RELATIONAL DATA FILE: REMARKS ON THE ADMISSIBILITY OF INPUT QUERIES (REF-DIR
GRCSI70 CIFICATION AND INTERROGATION OF FORMAL CONTROL SYSTEMS (REF-TEDAR71).=SPE
AQLNA65 ICAL THEORY OF INTERROGATIVES PART I, ANALYSIS (REF-KUJQQ71).=A NEW APPROACH TO THE LOG
OVCIN70 PROJECT INTREX SEMIANNUAL ACTIVITY REPORT (REF-MUJDC71).=
INTRX70 PROJECT INTREX SEMIANNUAL ACTIVITY REPORT (REF-KAAEU71) .=
MKRS71 INTRODUCTION AND PERSPECTIVES FOR THE 1971 ACM INFORMATION STORAGE AND RETRIEVAL SYMPOSIUM
MEEIM64 INTRODUCTION TO MATHEMATICAL LOGIC (REF-DIRRD71).=
LODIE68 AN INTRODUCTION TO ELF - LIST PROCESSING IN PL/I (REF-LODCC71).=
FEWIP68 AN INTRODUCTION TO PROBABILITY THEORY AND ITS APPLICATIONS (REF-GURER71).=
HAUIC67 INTRODUCTION TO COMPUTATIONAL LINGUISTICS (REF-GURER71).=
CARIS58 INTRODUCTION TO SYMBOLIC LOGIC AND ITS APPLICATIONS (REF-KUJQQ71).=
WILPI53 PHILOSOPHICAL INVESTIGATIONS (REF-PAMFS71).=
YNCMI62 COMIT AS AN IR LANGUAGE (REF-MKRS71).=
KAJRI67 RECENT ISSUES IN SEMANTIC THEORY (REF-PAMFS71).=

```
CARIS58        OLIC LOGIC AND      ITS                 APPLICATIONS (REF-KUJQQ71).=INTRODUCTION TO SYMB
FRAMR68        ODS APPLIED TO      JEWISH              CASE LAW (REF-CHYFT71).=MODERN RETRIEVAL METH
AFICP69        PS 1969 SPRING      JOINT               COMPUTER CONFERENCE (REF-MKRS71).=AFI
REACH64        TENCY OF HUMAN      JUDGEMENTS          OF RELEVANCE (REF-JADOC71).=THE CONSIS
TAMNU66           NON-UNIFORM      KEY                 DISTRIBUTION AND ADDRESS CALCULATION SORTING (REF-GURER71).=
TOJAA66        R CLUSTERED         KEYS                 (REF-GURER71).=ANALYSIS OF THE ADDRESS ASSIGNMENT PROBLEM FO
SCGMK63          A METHOD FOR      KEY-TO-ADDRESS      TRANSFORMATION (REF-GURER71).=
LUHKW59                            KEY-WORD-IN-CONTEXT IDEA FOR TECHNICAL LITERATURE (KWIC INDEX). (REF-MKRS71).=
HUECS70        IENCE TEACHING      LABORATORY          AT THE UNIVERSITY OF WASHINGTON (REF-HUEHM71).=THE COMPUTER SC
YNCMI62        COMIT AS AN IR      LANGUAGE             (REF-MKRS71).=COMIT
KECCN71        NVERSE NATURAL      LANGUAGE            DATA MANAGEMENT SYSTEM: CURRENT STATUS AND PLANS.=THE CO
WEJEC66        UDY OF NATURAL      LANGUAGE             COMMUNICATION BETWEEN MAN AND MACHINE (REF-LODCC71).=ELIZA:A COMPUTER PROGRAM
WOWTN70        RS FOR NATURAL      LANGUAGE            ANALYSIS (REF-KECCN71) (REF-DOBHF71) .=TRANSITION NETWORK GRAMMA
PAMFS71        S IN AUTOMATED      LANGUAGE            PROCESSING.=THE FUNCTION OF SEMANTIC
THFRE69        DLY EXTENSIBLE      LANGUAGE            SYSTEM (REF-DOBHF71).=REL: A RAPI
QURTL69         THE TEACHABLE      LANGUAGE            COMPREHENDER (REF-PAMFS71) (REF-SUSMS71) .=
SCRTC69        ION OF NATURAL      LANGUAGE             SEMANTICS (REF-PAMFS71).=TOWARDS A COMPUTATIONAL FORMALIZAT
SIRNL70              NATURAL       LANGUAGE            QUESTION ANSWERING SYSTEMS (REF-PAMFS71).=
SIMNL70              NATURAL       LANGUAGE            QUESTION-ANSWERING SYSTEMS. (REF-SUSMS71) (REF-MKRS71) .=
SAGAL68           AUTOMATED        LANGUAGE            PROCESSING (REF-PAMFS71).=
SCELE67                  A         LANGUAGE            ELEMENT FOR COMPRESSION CODING (REF-MUJDC71).=
KAJPL66         PHILOSOPHY OF      LANGUAGE             (REF-KECCN71).=THE
KEGNL68            A NATURAL       LANGUAGE            COMPILER FOR ON-LINE DATA MANAGEMENT (REF-SUSMS71) .=
MOCLA69         AND AUTOMATED      LANGUAGE            PROCESSING (REF-PAMFS71).=LINGUISTICS
LOCMS66        DEACON NATURAL      LANGUAGE             QUESTION-ANSWERING SYSTEM (REF-SUSMS71).=MEMORY STRUCTURES IN
KECNL68            A NATURAL       LANGUAGE            COMPILER FOR ONLINE DATA MANAGEMENT (REF-KECCN71).=
KECNL68            A NATURAL       LANGUAGE            COMPILER FOR ON-LINE DATA MANAGEMENT (REF-PAMFS71).=
LOPRE69        DLY EXTENSIBLE      LANGUAGE            SYSTEM: THE REL LANGUAGE PROCESSOR (REF-DOBHF71).=A RAPI
LOPRE69        DLY EXTENSIBLE      LANGUAGE            SYSTEM (REF-SUSMS71).=A RAPI
MOJNL69        AL AND NATURAL      LANGUAGE             (REF-PAMFS71) .=INFORMATION RETRIEV
KIGAL70        APL/B5500: THE      LANGUAGE            AND ITS IMPLEMENTATION (REF-HUEHM71).=APL/B5
MOCAL69             AUTOMATED      LANGUAGE            PROCESSING (REF-PAMFS71).=
LOPRE69        YSTEM: THE REL      LANGUAGE             PROCESSOR (REF-DOBHF71).=A RAPIDLY EXTENSIBLE LANGUAGE S
COLOL68        M WITH NATURAL      LANGUAGE             AND PICTORIAL INPUT (REF-SUSMS71).=AN ON-LINE QUESTION-ANSWERING SYSTE
DOGRE69        DLY EXTENSIBLE      LANGUAGE            SYSTEM: REL ENGLISH (REF-DOBHF71).=A RAPI
GAPLV63        ISTICS VIEW OF      LANGUAGE            DATA PROCESSING (REF-PAMFS71).=A LINGU
IBMPR67        ICTIVE NATURAL      LANGUAGE            SYSTEM (REF-LODCC71).=PROTO-RELADES:A RESTR
DEAPS65        ARGETING QUERY      LANGUAGE             (REF-LODCC71).=PHRASE-STRUCTURE ORIENTED T
HOCPU66         UNIVERSALS IN      LANGUAGE             (REF-PAMFS71).=THE PROBLEM OF
GAPNL63              NATURAL       LANGUAGE            AND THE COMPUTER (REF-GURER71).=
ALWPL64         PHILOSOPHY OF      LANGUAGE             (REF-PAMFS71).=
CARLS37        ICAL SYNTAX OF      LANGUAGE             (REF-KUJQQ71).=THE LOG
CHWTR62        ' ENGLISH-LIKE      LANGUAGE             (REF-MKRS71).=TRANSLATION OF RETRIEVAL REQUESTS COUCHED IN A 'SEMIFORMAL
BODAL67             AUTOMATED      LANGUAGE            PROCESSING (REF-PAMFS71).=
OTMIL69           INTERMEDIATE     LANGUAGES           FOR AUTOMATIC LANGUAGES PROCESSING (REF-PAMFS71).=
OTMIL69         FOR AUTOMATIC      LANGUAGES           PROCESSING (REF-PAMFS71).=INTERMEDIATE LANGUAGES
MOJTU67        ING OF NATURAL      LANGUAGES           BY MACHINES (REF-LODCC71).=TOWARDS THE UNDERSTAND
MUJDC71         PROCESSING OF      LARGE               COMMERCIAL FILES.=DATA COMPRESSION TECHNIQUES FOR ECONOMIC
WRMML60         MECHANIZING A      LARGE               INDEX; APPENDIX: THE SOUNDEX CODE (REF-MUJDC71).=
MCWTA69        GANIZATION FOR      LARGE                SCALE INFORMATION PROCESSING SYSTEMS (REF-WACQE71).=TOWARD THE AUTOMATIC DESI
MALMF69        ERMINATION FOR      LARGE               ON-LINE DATA FILES (REF-WIAFS71).=A MODEL FOR FILE STRUCTURE DET
KIGEL69        EXPERIMENTS IN      LARGE               SCALE COMPUTER DIRECT ACCESS STORAGE MANIPULATION (REF-HUEHM71).=EXPERI
WIRCR62        RIEVAL OF CASE      LAW                  (REF-CHYFT71).=COMPUTER RET
NIGSP69         ATOMIC ENERGY      LAW                 BY COMPUTER (REF-CHYFT71).=THE STATUS PROJECT:SEARCHING
SCACS69        RIEVAL OF CASE      LAW                 WITHOUT INDEXING: THE HEBREW RESPONSA PROJECT (REF-CHYFT71).=COMPUTERIZED STOR
FRAMR68        TO JEWISH CASE      LAW                  (REF-CHYFT71).=MODERN RETRIEVAL METHODS APPLIED
HOJSS62        HING STATUTORY      LAW                 BY COMPUTER (REF-CHYFT71).=SEARC
```

HIDLR69	THE	LEADER	RETRIEVAL SYSTEM (REF-SUSMS71) .=
CHYFT71	RIEVAL: HEBREW	LEGAL	TEXTS (REPORT ON THE FIRST PHASE OF THE RESPONSA RETRIEVAL PROJECT).=FULL TEXT
KAMLP62	FOR MECHANIZED	LEGAL	AND PATENT SEARCHING CENTERS (REF-CHYFT71).=A LONG-RANGE PROGRAM
ELW CT63	AS A TOOL FOR	LEGAL	RESEARCH (REF-CHYFT71).=THE COMPUTER
FRAC169	OOLS IN HEBREW	LEGAL	RESEARCH (IN HEBREW) (REF-CHYFT71).=THE COMPUTER AND INFORMATION RETRIEVAL A
FRAL168		LEGAL	INFORMATION RETRIEVAL (REF-CHYFT71).=
KAHIC69	(CIRCULATED	LETTER)	(REF-DOBHF71).=
OVLLA69		LIBRARY	AUTOMATION:A CRITICAL REVIEW (REF-SUSMS71).=
KIGLC63	MATION AND THE	LIBRARY	OF CONGRESS (REF-MKRS71).=AUTO
KAAEU71	UTILIZATION OF	LIMITED	ACCESS ARCHIVAL STORAGE IN A TIME SHARED ENVIRONMENT.=EFFICIENT
KAJLG68	A	LINEAR	GEOGRAPHICAL CODE FOR MANAGEMENT INFORMATION SYSTEMS (REF-MUJDC71).=
CAJUP65	OF PITTSBURGH	LINEAR	FILE TANDEM SYSTEM (REF-GURER71).=UPLIFTS-UNIVERSITY
WAUCL69	COMPUTATIONAL	LINGUISTIC	TECHNIQUES IN AN ON-LINE SYSTEM FOR TEXTUAL ANALYSIS (REF-SUSMS71).=
SCRST67	ARD A MODEL OF	LINGUISTIC	PERFORMANCE (REF-PAMFS71).=STEPS TOW
SIRLA68		LINGUISTIC	ANALYSIS OF CONSTRUCTED RESPONSES IN CAI (REF-PAMFS71).=
MOCLA69		LINGUISTICS	AND AUTOMATED LANGUAGE PROCESSING (REF-PAMFS71).=
GAPLV63	A	LINGUISTICS	VIEW OF LANGUAGE DATA PROCESSING (REF-PAMFS71).=
HAUIC67	COMPUTATIONAL	LINGUISTICS	(REF-GURER71).=INTRODUCTION TO
HAZSL61	STRUCTURAL	LINGUISTICS	(REF-PAMFS71).=
LOUIE68	CTION TO ELF -	LIST	PROCESSING IN PL/I (REF-LODCC71).=AN INTRODU
LUHKW59	FOR TECHNICAL	LITERATURE	(KWIC INDEX). (REF-MKRS71).=KEY-WORD-IN-CONTEXT INDEX
DOLSR	ROAD MAPS FOR	LITERATURE	SEARCHERS (REF-JADOC71).=SEMANTIC
SHJML67	MATHEMATICAL	LOGIC	(REF-DIRRD71).=
REHES47	TS OF SYMBOLIC	LOGIC	(REF-KUJQQ71).=ELEMEN
MEEIM64	O MATHEMATICAL	LOGIC	(REF-DIRRD71).=INTRODUCTION T
FERDS69	F MATHEMATICAL	LOGIC	(REF-KUJQQ71).=DICTIONARY OF SYMBOLS O
CARIS58	ON TO SYMBOLIC	LOGIC	AND ITS APPLICATIONS (REF-KUJQQ71).=INTRODUCTI
KUJAQ67	BY COMPUTER: A	LOGICAL	STUDY (REF-KUJQQ71).=ANSWERING QUESTIONS
KUJLA69		LOGICAL	ASPECTS OF QUESTION-ANSWERING BY COMPUTER (REF-KUJQQ71).=
KUJAQ67	BY COMPUTER: A	LOGICAL	STUDY (REF-DIRRD71).=ANSWERING QUESTIONS
HAUCL63	MMUNICATION: A	LOGICAL	MODEL (REF-KUJQQ71).=CO
CARLS37	THE	LOGICAL	SYNTAX OF LANGUAGE, (REF-KUJQQ71).=
AQLNA65	PPROACH TO THE	LOGICAL	THEORY OF INTERROGATIVES, PART I, ANALYSIS (REF-KUJQQ71).=A NEW A
KAMLP62	A	LONG-RANGE	PROGRAM FOR MECHANIZED LEGAL AND PATENT SEARCHING CENTERS (REF-CHYFT71).=
SGHSA64	ON METHODS FOR	MACHANIZED	DOCUMENTATION. (REF-MKRS71) .=STATISTICAL ASSOCIATI
WOWPS68	TION-ANSWERING	MACHINE	(REF-PAMFS71) (REF-SUSMS71) .=PROCEDURAL SEMANTICS FOR A QUES
WEJEC66	ETWEEN MAN AND	MACHINE	(REF-LODCC71).=ELIZA:A COMPUTER PROGRAM FOR THE STUDY OF NATURAL LANGUAGE COM
LEUMT64	THE	MACHINE	TOOLS OF COMBINATORICS (REF-GURER71).=
MOJTU67	L LANGUAGES BY	MACHINES	(REF-LODCC71).=TOWARDS THE UNDERSTANDING OF NATURA
GULEI63	ORMATION-LOGIC	MACHINES	(REF-GURER71).=ELECTRONIC INF
LOEDC68	LENDAR DATE FOR	MACHINE-TO-MACHINE	DATA INTERCHANGE (REF-MUJDC71).=DATA CODE FOR CA
KIJMA61	A	MAGNETIC	ASSOCIATIVE MEMORY (REF-GURER71).=
COJEM69	ABLISHMENT AND	MAINTENANCE	OF A STORAGE HIERARCHY FOR AN ON-LINE DATA BASE UNDER TSS/360 (REF-KAAEU71).=E
WEJEC66	CATION BETWEEN	MAN	AND MACHINE (REF-LODCC71).=ELIZA:A COMPUTER PROGRAM FOR THE STUDY OF NATURAL L
KECCN71	LANGUAGE DATA	MANAGEMENT	SYSTEM: CURRENT STATUS AND PLANS.=THE CONVERSE NATURAL
HUEHM71	BUILDING DATA	MANAGEMENT	PROGRAMS.=A HEATHKIT METHOD FOR
KEGNL68	R ON-LINE DATA	MANAGEMENT	(REF-SUSMS71) .=A NATURAL LANGUAGE COMPILER FO
KECDM68	DATA	MANAGEMENT	IN ORDINARY ENGLISH: EXAMPLES (REF-KECCN71).=
KECNL68	R ON-LINE DATA	MANAGEMENT	(REF-PAMFS71).=A NATURAL LANGUAGE COMPILER FO
MKRGD69	NERALIZED DATA	MANAGEMENT	SYSTEMS - SOME PRESPECTIVES (REF-MKRS71).=GE
KAJLG68	HICAL CODE FOR	MANAGEMENT	INFORMATION SYSTEMS (REF-MUJDC71).=A LINEAR GEOGRAP
KECNL68	OR ONLINE DATA	MANAGEMENT	(REF-KECCN71).=A NATURAL LANGUAGE COMPILER F
FIJGF70	NERALIZED FILE	MANAGEMENT	SYSTEM (REF-HUEHM71).=A GE
UOGED69	EMENTS OF DATA	MANAGEMENT	SYSTEMS (REF-GURER71).=EL
EISP68	MOTE FILE	MANAGEMENT	SYSTEM, PHASE 1 (REF-WIAFS71).=SYSTEMS PROGRAMMING DOCUMENTATION, TABLES, RE
IBMDM70	DATA	MANAGEMENT	FACILITIES OF TSS/360 (REF-KAAEU71).=
UAGI68	N, REMOTE FILE	MANAGEMENT	SYSTEM, PHASE I (REF-WIAFS71).=GENERAL INFORMATIO

COUSG69	IZED DATA BASE	MANAGEMENT	SYSTEMS (REF-MKRS71) .=A SURVEY OF GENERAL
SUSMS71		MANAGING	SEMANTIC DATA IN AN ASSOCIATIVE NET.=
SAGDM66	DATA	MANIPULATION	AND PROGRAMMING PROBLEMS IN AUTOMATIC INFORMATION RETRIEVAL (REF-GURER71).=
KIGEL69	CESS STORAGE	MANIPULATION	(REF-HUEHM71).=EXPERIMENTS IN LARGE SCALE COMPUTER DIRECT AC
MITEP64	H PREPROCESSOR	MANUAL	(REF-LODCC71).=MITRE, ENGLIS
IVKAU68	APL/360: USERS	MANUAL	(REF-HUEHM71).=APL/36
STISK63	SKETCHPAD, A	MAN-MACHINE	GRAPHICAL COMMUNICATION SYSTEM (REF-HUEHM71) .=
MECMM70		MAN-MACHINE	COMMUNICATIONS (REF-GURER71).=
DOLSR	SEMANTIC ROAD	MAPS	FOR LITERATURE SEARCHERS (REF-JADOC71).=
SLMMS69		MARRANOS	SEARCH IN THE RIVASH RESPONSA (IN HEBREW) (REF-CHYFT71).=
WHAPM50	PRINCIPIA	MATHEMATICA	(SECOND EDITION) (REF-KUJQQ71).=
SHJML67		MATHEMATICAL	LOGIC (REF-DIRRD71).=
MEEIM64	NTRODUCTION TO	MATHEMATICAL	LOGIC (REF-DIRRD71).=I
FERDS69	OF SYMBOLS OF	MATHEMATICAL	LOGIC (REF-KUJQQ71).=DICTIONARY
HAJMT67	OF SIMILARITY	MATRICES	BY TREES (REF-JADOC71).=REPRESENTATION
RUBIM62	N INQUIRY INTO	MEANING	AND TRUTH (REF-KUJQQ71).=A
SUSST69	ON INTERACTIVE	MEANING	(REF-SUSMS71).=A SEMANTIC THEORY BASED UP
BODAM65	ATOMIZATION OF	MEANING	(REF-PAMFS71).=THE
CARMN56		MEANING	AND NECESSITY (REF-KUJQQ71) (REF-KECCN71) .=
ABRIM69	FOR EXTRACTING	MEANING	FROM INPUT SENTENCES (REF-PAMFS71).=IMPLICATIONAL MOLECULES: A METHOU
ABRIM69	FOR EXTRACTING	MEANING	FROM INPUT SENTENCES (REF-SUSMS71).=IMPLICATIONAL MOLECULES: A METHOU
JAUBI69	IMPROVABILITY	MEASURE	IN INFORMATION RETRIEVAL (REF-JADOC71).=BASIS FOR AN
CESMT		MECHANICAL	TRANSLATION: THE CORRELATIONAL SOLUTION (REF-PAMFS71).=
LERTM65	LE: A TOOL FOR	MECHANIZED	INFERENCE EXECUTION AND DATA RETRIEVAL (REF-KECCN71).=RELATIONAL DATA FI
KEATM62	TEXTBOOK ON	MECHANIZED	INFORMATION RETRIEVAL (REF-GURER71).=
LERRD65	LE: A TOOL FOR	MECHANIZED	INFERENCE EXECUTION AND DATA RETRIEVAL (REF-KUJQQ71).=RELATIONAL DATA FI
KAMLP62	GE PROGRAM FOR	MECHANIZED	LEGAL AND PATENT SEARCHING CENTERS (REF-CHYFT71).=A LONG-RAN
WRMML60		MECHANIZING	A LARGE INDEX; APPENDIX: THE SOUNDEX CODE (REF-MUJUC71).=
PRAAP69	PROCESSING OF	MEDICAL	ENGLISH (REF-PAMFS71).=AUTOMATIC
PRAIT69	MORPHEMES IN	MEDICAL	ENGLISH (REF-PAMFS71).=IDENTIFICATION AND TRANSFORMATION OF TERMINAL
AUCME68		MEDLARS	1963-1967 (REF-SUSMS71).=
AUSTM64	THE	MEDLARS	SYSTEM, (REF-MKRS71).=
GURSM68	SEMANTIC	MEMORY	(REF-KECCN71) (REF-PAMFS71) .=
KIJMA61	IC ASSOCIATIVE	MEMORY	(REF-GURER71).=A MAGNET
LOCMS66		MEMORY	STRUCTURES IN DEACON NATURAL LANGUAGE QUESTION-ANSWERING SYSTEM (REF-SUSMS71).
CHBSH63	OF ASSOCIATIVE	MEMORY	(REF-GURER71).=SHIEF: A REALIZABLE FORM
HUEHM71	A FEATHKIT	METHOD	FOR BUILDING DATA MANAGEMENT PROGRAMS.=
SCGMK63	A	METHOD	FOR KEY-TO-ADDRESS TRANSFORMATION (REF-GURER71).=
GRMSG63	DECOMPOSITION	METHOD	(REF-TEDAR71).=SYSTEMS GENERATION OUTPUT
HUDMC	A	METHOD	FOR THE CONSTRUCTION OF MINIMUM REDUNDANCY CODES (REF-KAAEU71).=
ABRIM69	L MOLECULES: A	METHOD	FOR EXTRACTING MEANING FROM INPUT SENTENCES (REF-PAMFS71).=IMPLICATIONA
ABRIM69	L MOLECULES: A	METHOD	FOR EXTRACTING MEANING FROM INPUT SENTENCES (REF-SUSMS71).=IMPLICATIONA
BOAEN60	IENCY OF A NEW	METHOD	OF DICTIONARY CONSTRUCTION (REF-KNGHT71).=ON THE EFFIC
TEUMD69		METHODOLOGY	FOR THE DESIGN OF INFORMATION PROCESSING SYSTEMS (REF-TEDAR71).=
WIAMC70	A	METHODOLOGY	FOR COMPARISON OF FILE ORGANIZATION AND PROCESSING PROCEDURES FOR HIERARCHICAL
SWJEI67	TION RETRIEVAL	METHODS	(REF-JADOC71).=EFFECTIVENESS OF INFORMA
SGHSA64	AL ASSOCIATION	METHODS	FOR MACHANIZED DOCUMENTATION, (REF-MKRS71) .=STATISTIC
FRAMR68	DERN RETRIEVAL	METHODS	APPLIED TO JEWISH CASE LAW (REF-CHYFT71).=MO
BOCMI63		METHODS	OF INFORMATION HANDLING (REF-GURER71).=
KAMMS70	THE	MIND	SYSTEM: THE STRUCTURE OF THE SEMANTIC FILE (REF-SUSMS71).=
SCEDM63	DICTIONARY FOR	MINIMAL	REDUNDANCY ENCODING (REF-MUJDC71).=A
HUDMC	ONSTRUCTION OF	MINIMUM	REDUNDANCY CODES (REF-KAAEU71).=A METHOD FOR THE C
MITEP64		MITRE	ENGLISH PREPROCESSOR MANUAL (REF-LODCC71).=
SIRCM68	COMPUTATIONAL	MODEL	OF VERBAL UNDERSTANDING (REF-SUSMS71).=A
SIRCM68	COMPUTATIONAL	MODEL	OF VERBAL UNDERSTANDING (REF-PAMFS71).=A
SCRST67	STEPS TOWARD A	MODEL	OF LINGUISTIC PERFORMANCE (REF-PAMFS71).=STEPS

MALMF69	A	MODEL	FOR FILE STRUCTURE DETERMINATION FOR LARGE ON-LINE DATA FILES (REF-WIAFS71).=
HAUCL63	ION: A LOGICAL	MODEL	(REF-KUJQQ71).=COMMUNICAT
SEMFO68	ION EVALUATION	MODEL-FOREM	(REF-WACQE71).=A FILE ORGANIZAT
FICTM66	TOWARD A	MODERN	THEORY OF CASE (REF-PAMFS71).=
FRAMR68		MODERN	RETRIEVAL METHODS APPLIED TO JEWISH CASE LAW (REF-CHYFT71).=
ABRIM69	IMPLICATIONAL	MOLECULES	A METHOD FOR EXTRACTING MEANING FROM INPUT SENTENCES (REF-SUSMS71).=
ABRIM69	IMPLICATIONAL	MOLECULES	A METHOD FOR EXTRACTING MEANING FROM INPUT SENTENCES (REF-PAMFS71).=
PRAIT69	ON OF TERMINAL	MORPHEMES	IN MEDICAL ENGLISH (REF-PAMFS71).=IDENTIFICATION AND TRANSFORMATI
CHPPM68	PROCEDURE FOR	MORPHOLOGICAL	ANALYSIS (REF-PAMFS71).=A
PAMCM67	COMPUTATIONAL	MORPHOLOGY	(REF-PAMFS71).=
DAOE65	TRACING FROM A	MULTIPLE	FILE DATA STORAGE SYSTEM : AN APPLICATION OF INTEGER PROGRAMMING (REF-WACQE71)
KUSMP63		MULTIPLE-PATH	SYNTACTIC ANALYZER (REF-LODCC71).=
GHSAF68	RECORDS WITH	MULTIPLE-VALUED	ATTRIBUTES (REF-GURER71).=APPLICATION OF FINITE GEOMETRY IN FILE ORGANIZATION
GLEMP70	THE	MULTISTORE	PARSER FOR HIERARCHICAL SYNTACTIC STRUCTURES (REF-PAMFS71).=
LUVMR70		MULTI-ATTRIBUTE	RETRIEVAL WITH COMBINED INDEXES (REF-GURER71).=
SNMMV70	THE	MYRIAD	VIRTUES OF TEXT COMPACTION (REF-MUJDC71) .=
BRWGP66	RMATION AT THE	NASA	SCIENTIFIC AND TECHNICAL INFORMATION FACILITY (REF-SUSMS71).=GUIDE TO THE PROC
KECPA70	POSED AMERICAN	NATIONAL	STANDARD IDENTIFICATION OF STATES,.....,FOR INFORMATION INTERCHANGE (REF-MUJDC
KECCN71	THE CONVERSE	NATURAL	LANGUAGE DATA MANAGEMENT SYSTEM: CURRENT STATUS AND PLANS.=
WOWTN70	K GRAMMARS FOR	NATURAL	LANGUAGE ANALYSIS (REF-KECCN71) (REF-DOBHF71) .=TRANSITION NETWOR
WEJEC66	R THE STUDY OF	NATURAL	LANGUAGE COMMUNICATION BETWEEN MAN AND MACHINE (REF-LODCC71).=ELIZA:A COMPUTER
LODCC71	OR RESTRICTED,	NATURAL	ENGLISH.=CUE: A CUSTOMIZED SYSTEM F
SIMNL70		NATURAL	LANGUAGE QUESTION-ANSWERING SYSTEMS. (REF-SUSMS71) (REF-MKRS71) .=
SCRTC69	RMALIZATION OF	NATURAL	LANGUAGE SEMANTICS (REF-PAMFS71).=TOWARDS A COMPUTATIONAL FO
SIRNL70		NATURAL	LANGUAGE QUESTION ANSWERING SYSTEMS (REF-PAMFS71).=
KECNL68	A	NATURAL	LANGUAGE COMPILER FOR ON-LINE DATA MANAGEMENT (REF-PAMFS71).=
KECNL68	A	NATURAL	LANGUAGE COMPILER FOR ONLINE DATA MANAGEMENT (REF-KECCN71).=
KEGNL68	A	NATURAL	LANGUAGE COMPILER FOR ON-LINE DATA MANAGEMENT (REF-SUSMS71) .=
LOCMS66	URES IN DEACON	NATURAL	LANGUAGE QUESTION-ANSWERING SYSTEM (REF-SUSMS71).=MEMORY STRUCT
MOJNL69	RETRIEVAL AND	NATURAL	LANGUAGE (REF-PAMFS71) .=INFORMATION
MOJTU67	DERSTANDING OF	NATURAL	LANGUAGES BY MACHINES (REF-LODCC71).=TOWARDS THE UN
IBMPR67	:A RESTRICTIVE	NATURAL	LANGUAGE SYSTEM (REF-LODCC71).=PROTO-RELADES
GAPNL63		NATURAL	LANGUAGE AND THE COMPUTER (REF-GURER71).=
COLOL68	NG SYSTEM WITH	NATURAL	LANGUAGE AND PICTORIAL INPUT (REF-SUSMS71).=AN ON-LINE QUESTION-ANSWERI
CARMN56	MEANING AND	NECESSITY	(REF-KUJQQ71) (REF-KECCN71) .=
SUSMS71	AN ASSOCIATIVE	NET	=MANAGING SEMANTIC DATA IN
SHSNS69	A	NET	STRUCTURE BASED RELATIONAL QUESTION ANSWERER: DESCRIPTION AND EXAMPLES (REF-SU
WOWTN70	TRANSITION	NETWORK	GRAMMARS FOR NATURAL LANGUAGE ANALYSIS (REF-KECCN71) (REF-DOBHF71) .=
AQLNA65	A	NEW	APPROACH TO THE LOGICAL THEORY OF INTERROGATIVES, PART I, ANALYSIS (REF-KUJQQ7
BOAEN60	FFICIENCY OF A	NEW	METHOD OF DICTIONARY CONSTRUCTION (REF-KNGBT71).=ON THE E
COLSN65	SYSTEMATIZED	NOMENCLATURE	OF PATHOLOGY (REF-PAMFS71).=
TAMNU66		NON-UNIFORM	KEY DISTRIBUTION AND ADDRESS CALCULATION SORTING (REF-GURER71).=
CHNRN	REMARKS ON	NORMALIZATION	(REF-PAMFS71) .=
TANP65	A	NOTE	ON THE PSEUDO-MATHEMATICS OF RELEVANCE (REF-JADOC71).=
JADNS69	A	NOTE	ON A SET OF FUNCTIONS FOR INFORMATION RETRIEVAL (REF-JADOC71).=
BEJOA69	O INFORMATION:	NSF	AS AN AID TO THE INDEXER/CATALOGER (REF-CHYFT71).=ON-LINE ACCESS T
VEJED	OR THE DECIMAL	NUMBER	SYSTEM (REF-MUJDC71).=ERROR DETECTING AND CORRECTING CODES F
HAUDT64	ATION AND SOME	OBSERVATIONS	(REF-PAMFS71).=DEPENDENCY THEORY: A FORMALIZ
OLYTD68	IONAL AIDS FOR	OBTAINING	A FORMAL DESCRIPTION OF ENGLISH (REF-PAMFS71).=TOWARD THE DEVELOPMENT OF COMPU
SLMOR70	THE	OCCULT	IN THE RIVASH RESPONSA (IN HEBREW) (REF-CHYFT71).=
KOAOU66	AN	OCCUPANCY	DISCIPLINE AND APPLICATIONS, (REF-GURER71).=

KECNL68 E COMPILER FOR ONLINE DATA MANAGEMENT (REF-KECCN71).=A NATURAL LANGUAG
WAUCL69 CHNIQUES IN AN ON-LINE SYSTEM FOR TEXTUAL ANALYSIS (REF-SUSMS71).=COMPUTATIONAL LINGUISTIC TE
LEMDR68 N OF A REVISED ON-LINE INFORMATION RETRIEVAL SYSTEM (REF-KAAEU71) .=DESIG
KECNL68 E COMPILER FOR ON-LINE DATA MANAGEMENT (REF-PAMFS71).=A NATURAL LANGUAG
MALMF69 TION FOR LARGE ON-LINE DATA FILES (REF-WIAFS71).=A MODEL FOR FILE STRUCTURE DETERMINA
LEUFS69 STRUCTURES FOR ON-LINE SYSTEMS (REF-WIAFS71).=FILE
LEUFS67 STRUCTURES FOR ON-LINE SYSTEMS (REF-GURER71).=FILE
LEUFS69 STRUCTURES FOR ON-LINE SYSTEMS (REF-HUEHM71).=FILE
KEGNL68 E COMPILER FOR ON-LINE DATA MANAGEMENT (REF-SUSMS71) .=A NATURAL LANGUAG
COLOL68 AN ON-LINE QUESTION-ANSWERING SYSTEM WITH NATURAL LANGUAGE AND PICTORIAL INPUT (REF-SUSMS
BEJOA69 ON-LINE ACCESS TO INFORMATION: NSF AS AN AID TO THE INDEXER/CATALOGER (REF-CHYFT71).=
COJEM69 ERARCHY FOR AN ON-LINE DATA BASE UNDER TSS/360 (REF-KAAEU71).=ESTABLISHMENT AND MAINTENANCE OF A STOR
FIJUG70 S GUIDE TO THE OPERATION OF AN INFORMATION STORAGE AND RETRIEVAL SYSTEM ON THE B5500 COMPUTER (REF-HUEH
BEAOP55 OPERATIONISM (REF-PAMFS71).=
CHBCT65 RIEVAL CENTER OPERATIONS (REF-GURER71).=COLLOQUIUM ON TECHNICAL PRECONDITIONS FOR RET
JADOC71 OPTIMAL CLASSIFICATION AND ITS CONSEQUENCES.=
DAOE65 ON OPTIMAL EXTRACING FROM A MULTIPLE FILE DATA STORAGE SYSTEM : AN APPLICATION OF INTEGER
NUJD061 THE DESIGN AND OPTIMIZATION OF INFORMATION PROCESSING SYSTEMS (REF-WACQE71).=ON
KNDOB70 OPTIMUM BINARY SEARCH TREES (REF-KNGBT71).=
KECDM68 MANAGEMENT IN ORDINARY ENGLISH: EXAMPLES (REF-KECCN71).=DATA
TEDAR71 SEARCH IN FILE ORGANIZATION =AN APPROACH TO RE
SEMF068 A FILE ORGANIZATION EVALUATION MODEL-FOREM (REF-WACQE71).=
SAGAI63 IC INFORMATION ORGANIZATION AND RETRIEVAL (REF-MKRS71) (REF-GURER71).=AUTOMAT
WIAMC70 ARISON OF FILE ORGANIZATION AND PROCESSING PROCEDURES FOR HIERARCHICAL STORAGE STRUCTURES (REF-WIAFS71).=A
MCWTA69 DESIGN OF DATA ORGANIZATION FOR LARGE SCALE INFORMATION PROCESSING SYSTEMS (REF-WACQE71).=TOWARD THE AUTOM
GHSAF68 OMETRY IN FILE ORGANIZATION FOR RECORDS WITH MULTIPLE-VALUED ATTRIBUTES (REF-GURER71).=APPLICATION OF FINI
GHSF069 FILE ORGANIZATION ON THE SELECTION OF RANDOM ACCESS INDEX POINTS FOR SEQUENTIAL FILES (REF-WACQ
DZBOS65 THE ORGANIZATION OF STRUCTURED FILES (REF-GURER71).=
AFCFF67 FORMATTED FILE ORGANIZATION TECHNIQUES (REF-WACQE71).=FORMAT
ADGAO ORITHM FOR THE ORGANIZATION OF INFORMATION (REF-KNGBT71).=AN ALG
CHNCF69 ARISON OF FILE ORGANIZATION TECHNIQUES (REF-WIAFS71).=A COMP
BUWF063 FILE ORGANIZATION AND ADDRESSING (REF-WACQE71).=
UEAPS65 RASE-STRUCTURE ORIENTED TARGETING QUERY LANGUAGE (REF-LODCC71).=PH
LASOS66 AN OUTLINE OF STRATIFICATIONAL GRAMMAR (REF-PAMFS71).=
BAYOC THE OUTLOOK FOR COMPUTATIONAL SEMANTICS (REF-PAMFS71).=
GRMSG63 EMS GENERATION OUTPUT DECOMPOSITION METHOD (REF-TEDAR71).=SYST
SUSDR69 IRECTED RANDOM PARAGRAPH GENERATOR (REF-SUSMS71).=A D
KAMEP67 ITH A POWERFUL PARSER (REF-DOHHF71) (REF-KECCN71).=EXPERIMENTS W
GLEMP70 THE MULTISTORE PARSER FOR HIERARCHICAL SYNTACTIC STRUCTURES (REF-PAMFS71).=THE MU
AQLNA65 ROGATIVES. PART I, ANALYSIS (REF-KUJQQ71).=A NEW APPROACH TO THE LOGICAL THEORY OF INTER
EALAD67 TERMINATION OF PARTS OF SPEED OF ENGLISH WORDS (REF-PAMFS71).=AUTOMATIC DE
KAMLP62 IZED LEGAL AND PATENT SEARCHING CENTERS (REF-CHYFT71).=A LONG-RANGE PROGRAM FOR MECHAN
COLSN65 OMENCLATURE OF PATHOLOGY (REF-PAMFS71).=SYSTEMATIZED N
NAGSA68 OF THE ART IN PATTERN RECOGNITION (REF-JADOC71).=STATE
SCRST67 OF LINGUISTIC PERFORMANCE (REF-PAMFS71).=STEPS TOWARD A MODEL
CLCFD66 ETERMINING THE PERFORMANCE OF INDEXING SYSTEMS (REF-JADOC71).=FACTORS D
ARJIS69 ION SYSTEMS IN PERSPECTIVE (REF-GURER71).=INFORMAT
MKRS71 TRODUCTION AND PERSPECTIVES FOR THE 1971 ACM INFORMATION STORAGE AND RETRIEVAL SYMPOSIUM .=IN
CHYFT71 T ON THE FIRST PHASE OF THE RESPONSA RETRIEVAL PROJECT).=FULL TEXT DOCUMENT RETRIEVAL: HEBREW LEGAL
EISP68 GEMENT SYSTEM, PHASE 1 (REF-WIAFS71).=SYSTEMS PROGRAMMING DOCUMENTATION, TABLES, REMOTE FILE MANA
DAGI68 GEMENT SYSTEM, PHASE I (REF-WIAFS71).=GENERAL INFORMATION, REMOTE FILE MANA
COUIA62 MATION ALGEBRA PHASE I REPORT (REF-TEDAR71).=AN INFOR

```
WILPI53                                PHILOSOPHICAL     INVESTIGATIONS (REF-PAMFS71).=
KAJPL66                    THE         PHILOSOPHY        OF LANGUAGE (REF-KECCN71).=
MAMRD66        FILE I: DESIGN          PHILOSOPHY         (REF-DIRRD71).=RELATIONAL DATA
JEOPG65                    THE         PHILOSOPHY        OF GRAMMAR (REF-KUJQQ71).=
ALWPL64                                PHILOSOPHY        OF LANGUAGE (REF-PAMFS71).=
HAGGG              DEFENSE OF          PHRASE            STRUCTURE (REF-DOBHF71).=GENERATIVE GRAMMARS WITHOUT TRANSFORMATIONAL RULES: A
DEAPS65                                PHRASE-STRUCTURE  ORIENTED TARGETING QUERY LANGUAGE (REF-LODCC71).=
COLOL68         LANGUAGE  AND          PICTORIAL         INPUT (REF-SUSMS71).=AN ON-LINE QUESTION-ANSWERING SYSTEM WITH NATURAL
GAPPT67        TION TYPING: A          PILOT             STUDY IN SEMANTIC ANALYSIS; (REF-PAMFS71).=PREDICA
CAJUP65        -UNIVERSITY OF          PITTSBURGH        LINEAR FILE TANDEM SYSTEM (REF-GURER71).=UPLIFTS
OVCCI65                                PLANNING          CONFERENCE ON INFORMATION TRANSFER EXPERIMENTS (INTREX) (REF-MKRS71) .=
KECCN71        ENT STATUS AND          PLANS             =THE CONVERSE NATURAL LANGUAGE DATA MANAGEMENT SYSTEM: CURR
LOUIE68         PROCESSING IN          PL/I               (REF-LOUCC71).=AN INTRODUCTION TO ELF - LIST
GHSFO69         ACCESS INDEX           POINTS            FOR SEQUENTIAL FILES (REF-WACQE71) (REF-KNGBT71).=FILE ORGANIZATION: ON THE SE
KAMEP67        RIMENTS WITH A          POWERFUL          PARSER (REF-DOBHF71) (REF-KECCN71).=EXPE
CHBCT65        M ON TECHNICAL          PRECONDITIONS     FOR RETRIEVAL CENTER OPERATIONS (REF-GURER71).=COLLOQUIU
GAPPT67                                PREDICATION       TYPING: A PILOT STUDY IN SEMANTIC ANALYSIS; (REF-PAMFS71).=
BENAQ63         OF QUESTIONS:          PRELIMINARY       REPORT (REF-KUJQQ71).=AN ANALYSIS
MITEP64        MITRE, ENGLISH          PREPROCESSOR      MANUAL (REF-LODCC71).=MITRE,
MKRGU69        SYSTEMS - SOME          PRESPECTIVES       (REF-MKRS71).=GENERALIZED DATA MANAGEMENT
WHAPM50                                PRINCIPIA         MATHEMATICA (SECOND EDITION) (REF-KUJQQ71).=
GRCPI69                                PRINCIPLES        OF THE IDENTIFICATION OF INFORMATION (REF-TEDAR71).=
FEWIP68        NTRODUCTION TO          PROBABILITY       THEORY AND ITS APPLICATIONS (REF-GURER71).=AN I
DIRRD71        D THE DECISION          PROBLEM           FOR CLASSES OF PROPER FORMULAS.=THE RELATIONAL DATA FILE AN
TOJAA66        ESS ASSIGNMENT          PROBLEM           FOR CLUSTERED KEYS (REF-GURER71).=ANALYSIS OF THE ADDR
HOCPU66                    THE         PROBLEM           OF UNIVERSALS IN LANGUAGE (REF-PAMFS71).=
GRCAT69        REM PROVING TO          PROBLEM           SOLVING (REF-SUSMS71).=APPLICATION OF THEO
HAMAC63        A FILE ADDRESS          PROBLEM            (REF-GURER71).=AN APPLICATION OF CODING THEORY TO
DIRRD69        F THE DECISION          PROBLEM           FOR THE CLASS OF DEFINITE FORMULAS (REF-DIRRD71).=THE RECURSIVE UNSOLVABILITY
ACWSC54        F THE DECISION          PROBLEM            (REF-DIRRD71).=SOLVABLE CASES O
YOJAF59        ATA PROCESSING          PROBLEMS           (REF-TEDAR71).=ABSTRACT FORMULATION OF D
SAGDM66        ND PROGRAMMING          PROBLEMS          IN AUTOMATIC INFORMATION RETRIEVAL (REF-GURER71).=DATA MANIPULATION A
SAGEP70            EVALUATION          PROBLEMS          IN INTERACTIVE INFORMATION RETRIEVAL (REF-SUSMS71).=
EVTHP64        METRIC-ANALOGY          PROBLEMS           (REF-JAUOC71).=A HEURISTIC PROGRAM TO SOLVE GEO
CHECS69         CALCULUS WORD          PROBLEMS           (REF-SUSMS71).=COMPUTER SOLUTION OF
BOUQA64        LGEBRA WORK             PROBLEMS           (REF-KUJQQ71) (REF-SUSMS71) (REF-LODCC71).=A QUESTION-ANSWERING SYSTEM FOR HI

WOWPS68                                PROCEDURAL        SEMANTICS FOR A QUESTION-ANSWERING MACHINE (REF-PAMFS71) (REF-SUSMS71) .=
CHPPM68                    A           PROCEDURE         FOR MORPHOLOGICAL ANALYSIS (REF-PAMFS71).=
WIAMC70            PROCESSING          PROCEDURES        FOR HIERARCHICAL STORAGE STRUCTURES (REF-WIAFS71).=A METHODOLOGY FOR COMPARISO
SPUIS65        TEMS SCIENCES:          PROCEEDING        OF THE SECOND CONGRESS (REF-GURER71).=INFORMATION SYS
PAMFS71        MATED LANGUAGE          PROCESSING        =THE FUNCTION OF SEMANTICS IN AUTO
YOJAF59        LATION OF DATA          PROCESSING        PROBLEMS (REF-TEDAR71).=ABSTRACT FORMU
MUJDC71        S FOR ECONOMIC          PROCESSING        OF LARGE COMMERCIAL FILES.=DATA COMPRESSION TECHNIQUE
SAGCE68        EXING AND TEXT          PROCESSING         (REF-JAUOC71).=COMPUTER EVALUATION FOR IND
TEUMD69        OF INFORMATION          PROCESSING        SYSTEMS (REF-TEDAR71).=METHODOLOGY FOR THE DESIGN
SAGAL68        MATED LANGUAGE          PROCESSING         (REF-PAMFS71).=AUTO
```

WIAMC70	TION AND	PROCESSING	PROCEDURES FOR HIERARCHICAL STORAGE STRUCTURES (REF-WIAFS71).=A METHODOLOGY FO
TUSD63	EQUENTIAL DATA	PROCESSING	DESIGN (REF-TEDAR71).=S
PRAAP69	AUTOMATIC	PROCESSING	OF MEDICAL ENGLISH (REF-PAMFS71).=
LODIE68	TO ELF - LIST	PROCESSING	IN PL/1 (REF-LODCC71).=AN INTRODUCTION
CIMIL69	ATIC LANGUAGES	PROCESSING	(REF-PAMFS71).=INTERMEDIATE LANGUAGES FOR AUTOM
MOCLA69	MATED LANGUAGE	PROCESSING	(REF-PAMFS71).=LINGUISTICS AND AUTO
MUJD061	OF INFORMATION	PROCESSING	SYSTEMS (REF-WACQE71).=ON THE DESIGN AND OPTIMIZATION
MIMSI69	IC INFORMATION	PROCESSING	(REF-SUSMS71).=SEMANT
MOCAL69	MATED LANGUAGE	PROCESSING	(REF-PAMFS71).=AUTO
MCWTA69	LE INFORMATION	PROCESSING	SYSTEMS (REF-WACQE71).=TOWARD THE AUTOMATIC DESIGN OF DATA ORGANIZATION FOR LA
GAPLV63	LANGUAGE DATA	PROCESSING	(REF-PAMFS71).=A LINGUISTICS VIEW OF
BODAL67	MATED LANGUAGE	PROCESSING	(REF-PAMFS71).=AUTO
BRWGP66	GUIDE TO THE	PROCESSING	STORAGE AND RETRIEVAL OF BIBLIOGRAPHIC INFORMATION AT THE NASA SCIENTIFIC AND
LOPRE69	LANGUAGE	PROCESSOR	(REF-COMHF71).=A RAPIDLY EXTENSIBLE LANGUAGE SYSTEM: THE REL
FEJAB69	ASED ASSOCIATE	PROCESSOR	(REF-KECCN71).=AN ALGOL-B
ASWTR68	VE ASSOCIATIVE	PROCESSOR	WITH DEDUCTIVE CAPABILITIES (REF-KECCN71).=TRAMP: AN INTERPRETI
WEJEC66	IZA:A COMPUTER	PROGRAM	FOR THE STUDY OF NATURAL LANGUAGE COMMUNICATION BETWEEN MAN AND MACHINE (REF-L
THJPS69	A	PROGRAM	FOR THE SYNTACTIC ANALYSIS OF ENGLISH SENTENCES (REF-PAMFS71).=
EVTHP64	A HEURISTIC	PROGRAM	TO SOLVE GEOMETRIC-ANALOGY PROBLEMS (REF-JADOC71).=
KAMLP62	A LONG-RANGE	PROGRAM	FOR MECHANIZED LEGAL AND PATENT SEARCHING CENTERS (REF-CHYFT71).=
SAGDM66	NIPULATION AND	PROGRAMMING	PROBLEMS IN AUTOMATIC INFORMATION RETRIEVAL (REF-GURER71).=DATA MA
KNRAC69	RT OF COMPUTER	PROGRAMMING	VOL. 1 (REF-HUEHM71).=THE A
EISP68	SYSTEMS	PROGRAMMING	DOCUMENTATION, TABLES, REMOTE FILE MANAGEMENT SYSTEM, PHASE 1 (REF-WIAFS71).=
KNUAC68	RT OF COMPUTER	PROGRAMMING	VOL 1: FUNDAMENTAL ALGORITHMS (REF-KNGBT71).=THE A
GOSBP65	BACKTRACK	PROGRAMMING	(REF-GURER71).=
DAGE65	ION OF INTEGER	PROGRAMMING	(REF-WACQE71).=ON OPTIMAL EXTRACING FROM A MULTIPLE FILE DATA STORAGE SYSTEM
HUEHM71	ATA MANAGEMENT	PROGRAMS	=A HEATHKIT METHOD FOR BUILDING D
CHTGI68	TIONS ANALYSIS	PROGRAMS	FOR CONTEXT FREE GRAMMAR (REF-LODCC71).=ON THE GENERATION AND IMPLEMENTATION O
MUJPR69	A	PROGRESS	REPORT ON THE USE OF ENGLISH IN INFORMATION RETRIEVAL (REF-PAMFS71) .=
SCACS69	EBREW RESPONSA	PROJECT	(REF-CHYFT71).=COMPUTERIZED STORAGE AND RETRIEVAL OF CASE LAW WITHOUT INDEXIN
OVCIN70		PROJECT	INTREX SEMIANNUAL ACTIVITY REPORT (REF-MUJDC71).=
NIGSP69	THE STATUS	PROJECT	SEARCHING ATOMIC ENERGY LAW BY COMPUTER (REF-CHYFT71).=
INTRX70		PROJECT	INTREX SEMIANNUAL ACTIVITY REPORT (REF-KAAEU71) .=
CHYFT71	ONSA RETRIEVAL	PROJECT)	=FULL TEXT DOCUMENT RETRIEVAL: HEBREW LEGAL TEXTS (REPORT ON THE FIRST PHASE O
DIRRD71	CLASSES OF	PROPER	FORMULAS.=THE RELATIONAL DATA FILE AND THE DECISION PROBLEM FOR
HITSC62	COMBINATORIAL	PROPERTIES	OF CERTAIN TREES (REF-KNGBT71).=SOME
KECPA70		PROPOSED	AMERICAN NATIONAL STANDARD IDENTIFICATION OF STATES.....FOR INFORMATION INTE
LERRD69	M FOR	PROPOSITIONAL	DATA STORAGE AND INFERENCE EXEC (REF-DIRRD71).=RELATIONAL DATA FILE: EXPERIENC
IBMPR67		PROTO-RELAUES	A RESTRICTIVE NATURAL LANGUAGE SYSTEM (REF-LODCC71).=
GRCAT69	ION OF THEOREM	PROVING	TO PROBLEM SOLVING (REF-SUSMS71).=APPLICAT
TANP65	A NOTE ON THE	PSEUDO-MATHEMATICS	OF RELEVANCE (REF-JADOC71).=
KUJQQ71		QUANTIFICATION	IN QUERY SYSTEMS.=
WACQE71		QUANTITATIVE	EVALUATION OF DESIGN TRADEOFFS IN FILE SYSTEMS .=
KUJIR70	ILITY OF INPUT	QUERIES	(REF-DIRRD71).=INTERROGATING A RELATIONAL DATA FILE: REMARKS ON THE ADMISSIB
KUJIR70	ILITY OF INPUT	QUERIES	(REF-KUJQQ71).=INTERROGATING A RELATIONAL DATA FILE: REMARKS ON THE ADMISSIB
KUJQQ71	NTIFICATION IN	QUERY	SYSTEMS.=QUA
UEAPS65	NTED TARGETING	QUERY	LANGUAGE (REF-LODCC71).=PHRASE-STRUCTURE ORIE

SHSNS69 SED RELATIONAL QUESTION ANSWERER: DESCRIPTION AND EXAMPLES (REF-SUSMS71).=A NET STRUCTURE BA
SIRNL70 TURAL LANGUAGE QUESTION ANSWERING SYSTEMS (REF-PAMES71).=NA
WOWSI67 ION OF ENGLISH QUESTIONS ON A STRUCTURED DATA BASE (REF-PAMES71).=SEMANTIC INTERPRETAT
SIRAE65 WERING ENGLISH QUESTIONS BY COMPUTER: A SURVEY (REF-MKRS71) (REF-PAMES71) (REF-SUSMS71) .=ANS
KUJAQ67 ANSWERING QUESTIONS BY COMPUTER: A LOGICAL STUDY (REF-DIRRD71).=
KOMAQ69 F ENGLISH-LIKE QUESTIONS ABOUT SIMPLE DIAGRAMS (REF-SUSMS71).=AUTOMATIC QUESTION-ANSWERING O
KUJAQ67 ANSWERING QUESTIONS BY COMPUTER: A LOGICAL STUDY (REF-KUJQQ71).=
BENAQ63 AN ANALYSIS OF QUESTIONS PRELIMINARY REPORT (REF-KUJQQ71).=AN ANA
GWCLB61 : AN AUTOMATIC QUESTION-ANSWERER (REF-MKRS71).=BASEBALL
WOWPS68 SEMANTICS FOR A QUESTION-ANSWERING MACHINE (REF-PAMES71) (REF-SUSMS71) .=PROCEDURAL
SIMNL70 ATURAL LANGUAGE QUESTION-ANSWERING SYSTEMS. (REF-SUSMS71) (REF-MKRS71) .=N
KOMAQ69 AUTOMATIC QUESTION-ANSWERING OF ENGLISH-LIKE QUESTIONS ABOUT SIMPLE DIAGRAMS (REF-SUSMS71).=
LOCMS66 QUESTION-ANSWERING SYSTEM (REF-SUSMS71).=MEMORY STRUCTURES IN DEACON NATURAL LANGUAGE
KUJLA69 ICAL ASPECTS OF QUESTION-ANSWERING BY COMPUTER (REF-KUJQQ71).=LOG
BOUQA64 A QUESTION-ANSWERING SYSTEM FOR HIGH SCHOOL ALGEBRA WORK PROBLEMS (REF-KUJQQ71) (REF-SUSMS71) (REF
COLOL68 AN ON-LINE QUESTION-ANSWERING SYSTEM WITH NATURAL LANGUAGE AND PICTORIAL INPUT (REF-SUSMS71).=
NUWAW RADC-TDR-62-13 VOL. 1 (REF-MUJDC71).=
SUSDR69 A DIRECTED RANDOM PARAGRAPH GENERATOR (REF-SUSMS71).=
GHSFO69 E SELECTION OF RANDOM ACCESS INDEX POINTS FOR SEQUENTIAL FILES (REF-WACUE71) (REF-KNGHT71).=FILE ORG
GURER71 LEMENTS OF THE RANDOMIZED COMBINATORIAL FILE STRUCTURE.=E
GSKRC69 A RANDOMIZED COMBINATORIAL FILE STRUCTURE FOR STORAGE AND RETRIEVAL SYSTEMS (REF-GURER71).=
PEWAR57 ADDRESSING FOR RANDOM-ACCESS STORAGE (REF-GURER71).=ADDRES
THFRE69 REL: A RAPIDLY EXTENSIBLE LANGUAGE SYSTEM (REF-DOBHF71).=
LOPRE69 A RAPIDLY EXTENSIBLE LANGUAGE SYSTEM (REF-SUSMS71).=
LOPRE69 A RAPIDLY EXTENSIBLE LANGUAGE SYSTEM: THE REL LANGUAGE PROCESSOR (REF-DOBHF71).=
UOGRE69 A RAPIDLY EXTENSIBLE LANGUAGE SYSTEM: REL ENGLISH (REF-DOBHF71).=
CHBSH63 SHIEF: A REALIZABLE FORM OF ASSOCIATIVE MEMORY (REF-GURER71).=
MAJDR67 DESIGN OF REAL-TIME COMPUTER SYSTEMS (REF-TEDAR71).=
WIPTF60 , FORESTS, AND REARRANGING (REF-KNGBT71).=TREES
KAJRI67 RECENT ISSUES IN SEMANTIC THEORY (REF-PAMES71).=
NAGSA68 ART IN PATTERN RECOGNITION (REF-JADOC71).=STATE OF THE
EAJGR65 GENERATING A RECOGNIZER FOR A BNF GRAMMAR (REF-DOBHF71).=
CUPAR69 AN AUTOMATED RECOGNTION GRAMMAR FOR ENGLISH (REF-LODCC71).=
GHSAF68 ZATION FOR RECORDS WITH MULTIPLE-VALUED ATTRIBUTES (REF-GURER71).=APPLICATION OF FINITE GEOMETRY
DIRRU69 THE RECURSIVE UNSOLVABILITY OF THE DECISION PROBLEM FOR THE CLASS OF DEFINITE FORMULAS (REF-
CHTG168 LEMENTATION OF REDUCTIONS ANALYSIS PROGRAMS FOR CONTEXT FREE GRAMMAR (REF-LOUCC71).=ON THE GENERATION AN
SCEDM63 RY FOR MINIMAL REDUNDANCY ENCODING (REF-MUJDC71).=A DICTIONA
HUUMC ION OF MINIMUM REDUNDANCY CODES (REF-KAAEU71).=A METHOD FOR THE CONSTRUCT
THFRE69 REL A RAPIDLY EXTENSIBLE LANGUAGE SYSTEM (REF-DOBHF71).=
LOPRE69 GE SYSTEM: THE REL LANGUAGE PROCESSOR (REF-DOBHF71).=A RAPIDLY EXTENSIBLE LANGUA
UOGRE69 NGUAGE SYSTEM: REL ENGLISH (REF-DOBHF71).=A RAPIDLY EXTENSIBLE LA
DIRRD71 THE RELATIONAL DATA FILE AND THE DECISION PROBLEM FOR CLASSES OF PROPER FORMULAS.=
WEPCR71 CYLINDERS: A RELATIONAL DATA STRUCTURE (REF-SUSMS71).=
SHSNS69 TRUCTURE BASED RELATIONAL QUESTION ANSWERER: DESCRIPTION AND EXAMPLES (REF-SUSMS71).=A NET S
LERRD66 RELATIONAL DATA FILE II: IMPLEMENTATION (REF-DIRRD71).=
LERTM65 RELATIONAL DATA FILE: A TOOL FOR MECHANIZED INFERENCE EXECUTION AND DATA RETRIEVAL (REF-K
MIJRD70 RELATIONAL DATA SYSTEM STUDY (REF-MKRS71) .=
LERRD65 RELATIONAL DATA FILE: A TOOL FOR MECHANIZED INFERENCE EXECUTION AND DATA RETRIEVAL (REF-K
KUJIR70 NTERROGATING A RELATIONAL DATA FILE: REMARKS ON THE ADMISSIBILITY OF INPUT QUERIES (REF-DIRRD71).=I
LERRD69 RELATIONAL DATA FILE: EXPERIENCE WITH A SYSTEM FOR PROPOSITIONAL DATA STORAGE AND INFEREN
KUJIR70 NTERROGATING A RELATIONAL DATA FILE: REMARKS ON THE ADMISSIBILITY OF INPUT QUERIES (REF-KUJQQ71).=I
MAMRD66 RELATIONAL DATA FILE I: DESIGN PHILOSOPHY (REF-DIRRD71).=
TAACR41 HE CALCULUS OF RELATIONS (REF-KUJQQ71).=ON T
REACH64 JUDGEMENTS OF RELEVANCE (REF-JADOC71).=THE CONSISTENCY OF HUMAN
SAGRA69 RELEVANCE ASSESSMENTS AND RETRIEVAL SYSTEM EVALUATION (REF-JADOC71).=
TANP65 MATHEMATICS OF RELEVANCE (REF-JADOC71).=A NOTE ON THE PSEUDO-
JADCR LASSIFICATION, RELEVANCE AND INFORMATION RETRIEVAL (REF-JADOC71).=C

```
KUJIR70   NAL DATA FILE:   REMARKS              ON THE ADMISSIBILITY OF INPUT QUERIES (REF-DIRRD71).=INTERROGATING A RELATIO
KUJIR70   NAL DATA FILE:   REMARKS              ON THE ADMISSIBILITY OF INPUT QUERIES (REF-KUJQQ71).=INTERROGATING A RELATIO
CHNRN                      REMARKS              ON NORMALIZATION (REF-PAMFS71) .=
DAGI68    L INFORMATION,   REMOTE               FILE MANAGEMENT SYSTEM, PHASE I (REF-WIAFS71).=GENERA
EISP68    ATION, TABLES,   REMOTE               FILE MANAGEMENT SYSTEM, PHASE 1 (REF-WIAFS71).=SYSTEMS PROGRAMMING DOCUMENT
STMAI70   ATE OF THE ART   REPORT               (REF-MKRS71).=AUTOMATIC INDEXING: A ST
MOJPR69       A PROGRESS   REPORT               ON THE USE OF ENGLISH IN INFORMATION RETRIEVAL (REF-PAMFS71) .=
OVCIN70   NNUAL ACTIVITY   REPORT               (REF-MUJDC71).=PROJECT INTREX SEMIA
INTRX70   NNUAL ACTIVITY   REPORT               (REF-KAAEU71) .=PROJECT INTREX SEMIA
COUIA62   LGEBRA PHASE I   REPORT               (REF-TEUAR71).=AN INFORMATION A
BENAQ63   S: PRELIMINARY   REPORT               (REF-KUJQQ71).=AN ANALYSIS OF QUESTION
HAJMT67                    REPRESENTATION       OF SIMILARITY MATRICES BY TREES (REF-JADOC71).=
CHWTR62   N OF RETRIEVAL   REQUESTS             COUCHED IN A 'SEMIFORMAL' ENGLISH-LIKE LANGUAGE. (REF-MKRS71).=TRANSLATIO
TEUAR71   AN APPROACH TO   RESEARCH             IN FILE ORGANIZATION.=AN APP
FRACI69   N HEBREW LEGAL   RESEARCH             ( IN HEBREW ) (REF-CHYFT71).=THE COMPUTER AND INFORMATION RETRIEVAL AS AUXILIA
ELW CT63  TOOL FOR LEGAL   RESEARCH             (REF-CHYFT71).=THE COMPUTER AS A
DOBHF71     HOW FEATURES   RESOLVE              SYNTACTIC AMBIGUITY.=
CHYFT71   T PHASE OF THE   RESPONSA             RETRIEVAL PROJECT).=FULL TEXT DOCUMENT RETRIEVAL: HEBREW LEGAL TEXTS (REPORT O
SCACS69   NG: THE HEBREW   RESPONSA             PROJECT (REF-CHYFT71).=COMPUTERIZED STORAGE AND RETRIEVAL OF CASE LAW WITHOUT
SLMOR70    IN THE RIVASH   RESPONSA             (IN HEBREW) (REF-CHYFT71).=THE OCCULT
SLMMS69    IN THE RIVASH   RESPONSA             (IN HEBREW) (REF-CHYFT71).=MARRANOS SEARCH
SIRLA68   OF CONSTRUCTED   RESPONSES            IN CAI (REF-PAMFS71).=LINGUISTIC ANALYSIS
LOUCC71   ZED SYSTEM FOR   RESTRICTED           NATURAL ENGLISH.=CUE: A CUSTOMI
IBMPR67   ROTO-RELADES:A   RESTRICTIVE          NATURAL LANGUAGE SYSTEM (REF-LOUCC71).=P
CHYFT71    TEXT DOCUMENT   RETRIEVAL            HEBREW LEGAL TEXTS (REPORT ON THE FIRST PHASE OF THE RESPONSA RETRIEVAL PROJE
KNGBT71   EE STORAGE AND   RETRIEVAL            ALGORITHM.=A BALANCED TR
CHYFT71   F THE RESPONSA   RETRIEVAL            PROJECT).=FULL TEXT DOCUMENT RETRIEVAL: HEBREW LEGAL TEXTS (REPORT ON THE FIRS
MKRS71    N  STORAGE AND   RETRIEVAL            SYMPOSIUM .=INTRODUCTION AND PERSPECTIVES FOR THE 1971 ACM INFORMATIO
SAGDM66      INFORMATION   RETRIEVAL            (REF-GURER71).=DATA MANIPULATION AND PROGRAMMING PROBLEMS IN AUTOMATIC
SAGEP70   VE INFORMATION   RETRIEVAL            (REF-SUSMS71).=EVALUATION PROBLEMS IN INTERACTI
SAGAI63   GANIZATION AND   RETRIEVAL            (REF-MKRS71) (REF-GURER71).=AUTOMATIC INFORMATION OR
SWJEI67   OF INFORMATION   RETRIEVAL            METHODS (REF-JADOC71).=EFFECTIVENESS
SCACS69   ED STORAGE AND   RETRIEVAL            OF CASE LAW WITHOUT INDEXING: THE HEBREW RESPONSA PROJECT (REF-CHYFT71).=COMPU
VIBOR65             ON     RETRIEVAL            SYSTEM THEORY (REF-GURER71).=
WACIR65      INFORMATION   RETRIEVAL            (REF-GURER71).=
WIRCR62         COMPUTER   RETRIEVAL            OF CASE LAW (REF-CHYFT71).=
SAGSA65   MATIC DOCUMENT   RETRIEVAL            SYSTEM -- AN ILLUSTRATION (REF-JADOC71).=THE SMART AUTO
SAGRA69   SSESSMENTS AND   RETRIEVAL            SYSTEM EVALUATION (REF-JADOC71).=RELEVANCE A
SEMIS69   ON STORAGE AND   RETRIEVAL            SYSTEMS (REF-CHYFT71).=INFORMATI
SCG67        INFORMATION   RETRIEVAL            A CRITICAL VIEW (REF-GURER71).=
LEMDR68   NE INFORMATION   RETRIEVAL            SYSTEM (REF-KAAEU71) .=DESIGN OF A REVISED ON-LI
LERRD65   UTION AND DATA   RETRIEVAL            (REF-KUJQQ71).=RELATIONAL DATA FILE: A TOOL FOR MECHANIZED INFERENCE EXEC
LERCS67   UTION AND DATA   RETRIEVAL            (REF-KUJQQ71).=A COMPUTER SYSTEM FOR INFERENCE EXEC
NUWCW68   OR INFORMATION   RETRIEVAL            (REF-MUJDC71).=COMPRESSION WORD CODING TECHNIQUES F
LERTM65   UTION AND DATA   RETRIEVAL            (REF-KECCN71).=RELATIONAL DATA FILE: A TOOL FOR MECHANIZED INFERENCE EXEC
KEATM62   ED INFORMATION   RETRIEVAL            (REF-GURER71).=TEXTBOOK ON MECHANIZ
LUVMR70   ULTI-ATTRIBUTE   RETRIEVAL            WITH COMBINED INDEXES (REF-GURER71).=M
MOJPR69   FORMATION        RETRIEVAL            (REF-PAMFS71) .=A PROGRESS REPORT ON THE USE OF ENGLISH IN IN
LERCS67   UTION AND DATA   RETRIEVAL            (REF-DIRRD71).=A COMPUTER SYSTEM FOR INFERENCE EXEC
MOJNL69      INFORMATION   RETRIEVAL            AND NATURAL LANGUAGE (REF-PAMFS71) .=
JADNS69   OR INFORMATION   RETRIEVAL            (REF-JADOC71).=A NOTE ON A SET OF FUNCTIONS F
FRALI68   AL INFORMATION   RETRIEVAL            (REF-CHYFT71).=LEG
FRACI69   ND INFORMATION   RETRIEVAL            AS AUXILIARY TOOLS IN HEBREW LEGAL RESEARCH ( IN HEBREW ) (REF-CHYFT71).=THE C
JADCR     ND INFORMATION   RETRIEVAL            (REF-JADOC71).=CLASSIFICATION, RELEVANCE A
HIULR69       THE LEADER   RETRIEVAL            SYSTEM (REF-SUSMS71) .=
FOCIS65   ON STORAGE AND   RETRIEVAL            USING AVL TREES (REF-KNGBT71).=INFORMATI
GSRRC69    STORAGE AND     RETRIEVAL            SYSTEMS (REF-GURER71).=A RANDOMIZED COMBINATORIAL FILE STRUCTURE FOR
```

```
FIJUG70   ON STORAGE AND    RETRIEVAL   SYSTEM ON THE B5500 COMPUTER (REF-HUEHM71).=A USERS GUIDE TO THE OPERATION OF
FRAMR68         MODERN      RETRIEVAL   METHODS APPLIED TO JEWISH CASE LAW (REF-CHYFT71).=
JAUBI69   IN INFORMATION    RETRIEVAL    (REF-JADOC71).=BASIS FOR AN IMPROVABILITY MEASURE
HSUFS70   OR INFORMATION    RETRIEVAL   FROM FILES (REF-GURER71).=A FORMAL SYSTEM F
CHBCT65   CONDITIONS FOR    RETRIEVAL   CENTER OPERATIONS (REF-GURER71).=COLLOQUIUM ON TECHNICAL PRE
BRWGP66   G, STORAGE AND    RETRIEVAL   OF BIBLIOGRAPHIC INFORMATION AT THE NASA SCIENTIFIC AND TECHNICAL INFORMATION
COMMS70   XT INFORMATION    RETRIEVAL   ( IN HEBREW ) (REF-CHYFT71).=FULL TE
CORIS69   ON STORAGE AND    RETRIEVAL    (REF-SUSMS71).=INFORMATI
BEJIS63   ON STORAGE AND    RETRIEVAL    TOOLS, ELEMENTS, THEORIES (REF-GURER71).=INFORMATI
CHWTR62   TRANSLATION OF    RETRIEVAL   REQUESTS COUCHED IN A 'SEMIFORMAL' ENGLISH-LIKE LANGUAGE. (REF-MKRS71).=TRANSL
TEURS69               A     REVIEW    OF SYSTEM DESIGN TECHNIQUES (REF-WACOE71).=
OVLLA69   ION:A CRITICAL      REVIEW     (REF-SUSMS71).=LIBRARY AUTOMAT
LEMDR68     DESIGN OF A       REVISED   ON-LINE INFORMATION RETRIEVAL SYSTEM (REF-KAAEU71) .=
SLMMS69    SEARCH IN THE      RIVASH    RESPONSA (IN HEBREW) (REF-CHYFT71).=MARRANOS
SLMOR70    OCCULT IN THE      RIVASH    RESPONSA (IN HEBREW) (REF-CHYFT71).=THE
DOLSR           SEMANTIC      ROAD      MAPS FOR LITERATURE SEARCHERS (REF-JADOC71).=
HAGGG     ANSFORMATIONAL      RULES      A DEFENSE OF PHRASE STRUCTURE (REF-DOBHF71).=GENERATIVE GRAMMARS WITHOUT TR
IBMTA68    SYSTEM (TAG):      SALES     AND SYSTEMS GUIDE (REF-TEDAR71).=THE TIME AUTOMATED GRID
KIGEL69   MENTS IN LARGE      SCALE     COMPUTER DIRECT ACCESS STORAGE MANIPULATION (REF-HUEHM71).=EXPERI
MCWTA69   ION FOR LARGE       SCALE     INFORMATION PROCESSING SYSTEMS (REF-WACOE71).=TOWARD THE AUTOMATIC DESIGN OF D
BOUQA64   YSTEM FOR HIGH      SCHOOL    ALGEBRA WORK PROBLEMS (REF-KUJQQ71) (REF-SUSMS71) (REF-LOUCC71).=A QUESTION-AN
YOMIS69      INFORMATION      SCIENCE    TOWARD THE DEVELOPMENT OF A TRUE SCIENTIFIC DISCIPLINE (REF-SUSMS71).=
HUECS70    THE COMPUTER       SCIENCE   TEACHING LABORATORY AT THE UNIVERSITY OF WASHINGTON (REF-HUEHM71).=
SPJIS65   MATION SYSTEMS      SCIENCES    PROCEEDING OF THE SECOND CONGRESS (REF-GURER71).=INFOR
YOMIS69   F A TRUE            SCIENTIFIC DISCIPLINE (REF-SUSMS71).=INFORMATION SCIENCE: TOWARD THE DEVELOPMENT O
BRWGP66   ON AT THE NASA      SCIENTIFIC AND TECHNICAL INFORMATION FACILITY (REF-SUSMS71).=GUIDE TO THE PROCESSING, STO
SLMMS69        MARRANOS       SEARCH    IN THE RIVASH RESPONSA (IN HEBREW) (REF-CHYFT71).=
KNUOB70   OPTIMUM BINARY      SEARCH    TREES (REF-KNGBT71).=OPTIMU
COKES69   RIMENTS WITH A      SEARCH    ALGORITHM FOR THE DATA BASE OF A HUMAN BELIEF SYSTEM (REF-SUSMS71) (REF-PAMFS7
DOLSR     FOR LITERATURE      SEARCHERS  (REF-JADOC71).=SEMANTIC ROAD MAPS
NIGSP69   STATUS PROJECT:     SEARCHING ATOMIC ENERGY LAW BY COMPUTER (REF-CHYFT71).=THE
FEUBT68    BALANCED TREE      SEARCHING  (REF-KNGBT71).=
HOJSS62                       SEARCHING STATUTORY LAW BY COMPUTER (REF-CHYFT71).=
KAMLP62     PATENT            SEARCHING CENTERS (REF-CHYFT71).=A LONG-RANGE PROGRAM FOR MECHANIZED LEGAL AND
SPJIS65   CEEDING OF THE      SECOND     CONGRESS (REF-GURER71).=INFORMATION SYSTEMS SCIENCES: PRO
PRFSU39                       SECRET    AND URGENT, THE STORY OF CODES AND CIPHERS (REF-MUJDC71) .=
GHSFO69   ZATION: ON THE      SELECTION OF RANDOM ACCESS INDEX POINTS FOR SEQUENTIAL FILES (REF-WACOE71) (REF-KNGBT71)
SUSMS71         MANAGING      SEMANTIC  DATA IN AN ASSOCIATIVE NET.=
SUSST69               A       SEMANTIC  THEORY BASED UPON INTERACTIVE MEANING (REF-SUSMS71).=
WEUES66   XPLORATIONS IN      SEMANTIC  THEORY (REF-PAMFS71).=E
WOWSI67                       SEMANTIC  INTERPRETATION OF ENGLISH QUESTIONS ON A STRUCTURED DATA BASE (REF-PAMFS71).=
GURSM68                       SEMANTIC  MEMORY (REF-KECCN71) (REF-PAMFS71) .=
KAMMS70   RUCTURE OF THE      SEMANTIC  FILE (REF-SUSMS71).=THE MIND SYSTEM: THE ST
MIMSI69                       SEMANTIC  INFORMATION PROCESSING (REF-SUSMS71).=
GAPPT67   PILOT STUDY IN      SEMANTIC  ANALYSIS: (REF-PAMFS71).=PREDICATION TYPING: A
KAUSS65   STRUCTURE OF A      SEMANTIC  THEORY (REF-PAMFS71).=THE
KAUR167   CENT ISSUES IN      SEMANTIC  THEORY (REF-PAMFS71).=RE
DOLSR                         SEMANTIC  ROAD MAPS FOR LITERATURE SEARCHERS (REF-JADOC71).=
PAMFS71   HE FUNCTION OF      SEMANTICS IN AUTOMATED LANGUAGE PROCESSING.=T
SCRTC69   RAL LANGUAGE        SEMANTICS  (REF-PAMFS71).=TOWARDS A COMPUTATIONAL FORMALIZATION OF NATU
WOWPS68       PROCEDURAL      SEMANTICS FOR A QUESTION-ANSWERING MACHINE (REF-PAMFS71) (REF-SUSMS71) .=
BAYOC      COMPUTATIONAL      SEMANTICS  (REF-PAMFS71).=THE OUTLOOK FOR
OVCIN70   PROJECT INTREX      SEMIANNUAL ACTIVITY REPORT (REF-MUJDC71).=PROJEC
INTRX70   PROJECT INTREX      SEMIANNUAL ACTIVITY REPORT (REF-KAAEU71) .=PROJEC
THJPS69   SIS OF ENGLISH      SENTENCES  (REF-PAMFS71).=A PROGRAM FOR THE SYNTACTIC ANALY
ABRIM69       FROM INPUT      SENTENCES  (REF-PAMFS71).=IMPLICATIONAL MOLECULES: A METHOD FOR EXTRACTING MEANING
ABRIM69       FROM INPUT      SENTENCES  (REF-SUSMS71).=IMPLICATIONAL MOLECULES: A METHOD FOR EXTRACTING MEANING
```

```
TUSD63                          SEQUENTIAL      DATA PROCESSING DESIGN (REF-TEDAR71).=
GHSF069      EX  POINTS FOR     SEQUENTIAL      FILES (REF-WACOE71) (REF-KNGBT71).=FILE ORGANIZATION: ON THE SELECTION OF RAND
JADNS69        A NOTE ON A      SET             OF FUNCTIONS FOR INFORMATION RETRIEVAL (REF-JADOC71).=
KAAEU71      RAGE IN A TIME     SHARED          ENVIRONMENT.=EFFICIENT UTILIZATION OF LIMITED ACCESS ARCHIVAL STO
CHBSH63                         SHIEF            A REALIZABLE FORM OF ASSOCIATIVE MEMORY (REF-GURER71).=
HAJMT67      RESENTATION OF     SIMILARITY      MATRICES BY TREES (REF-JADOC71).=REP
KOMAQ69      UESTIONS ABOUT     SIMPLE          DIAGRAMS (REF-SUSMS71).=AUTOMATIC QUESTION-ANSWERING OF ENGLISH-LIKE Q
STISK63                         SKETCHPAD        A MAN-MACHINE GRAPHICAL COMMUNICATION SYSTEM (REF-HUEHM71) .=
SAGSA65                  THE    SMART           AUTOMATIC DOCUMENT RETRIEVAL SYSTEM -- AN ILLUSTRATION (REF-JADOC71).=
CESMT         CORRELATIONAL     SOLUTION         (REF-PAMFS71).=MECHANICAL TRANSLATION: THE
CHECS69            COMPUTER     SOLUTION        OF CALCULUS WORD PROBLEMS (REF-SUSMS71).=
ACWSC54                         SOLVABLE        CASES OF THE DECISION PROBLEM (REF-DIRRD71).=
EVTHP64      TIC PROGRAM TO     SOLVE           GEOMETRIC-ANALOGY PROBLEMS (REF-JADOC71).=A HEURIS
GRCAI69      ING TO PROBLEM     SOLVING          (REF-SUSMS71).=APPLICATION OF THEOREM PROV
TAMNU66      SS CALCULATION     SORTING          (REF-GURER71).=NON-UNIFORM KEY DISTRIBUTION AND ADDRE
FLIAI61      ERNAL COMPUTER     SORTING          (REF-WIAFS71).=ANALYSIS OF INT
WRMML60       APPENDIX: THE     SOUNDEX         CODE (REF-MUJDC71).=MECHANIZING A LARGE INDEX:
GRCSI70                         SPECIFICATION   AND INTERROGATION OF FORMAL CONTROL SYSTEMS (REF-TEDAR71).=
EALAD67      ON OF PARTS OF     SPEED           OF ENGLISH WORDS (REF-PAMFS71).=AUTOMATIC DETERMINATI
AFICP69          AFIPS 1969     SPRING          JOINT COMPUTER CONFERENCE (REF-MKRS71).=
KECPA70      RICAN NATIONAL     STANDARD        IDENTIFICATION OF STATES.......FOR INFORMATION INTERCHANGE (REF-MUJDC71).=PROP
STMAI70      IC INDEXING: A     STATE           OF THE ART REPORT (REF-MKRS71).=AUTOMAT
NAGSA68                         STATE           OF THE ART IN PATTERN RECOGNITION (REF-JADOC71).=
KECPA70      NTIFICATION OF     STATES          ......FOR INFORMATION INTERCHANGE (REF-MUJDC71).=PROPOSED AMERICAN NATIONAL ST
SGHSA64                         STATISTICAL     ASSOCIATION METHODS FOR MACHANIZED DOCUMENTATION. (REF-MKRS71) .=
KECCN71              CURRENT    STATUS          AND PLANS.=THE CONVERSE NATURAL LANGUAGE DATA MANAGEMENT SYSTEM:
NIGSP69                 THE     STATUS          PROJECT:SEARCHING ATOMIC ENERGY LAW BY COMPUTER (REF-CHYFT71).=
HOJSS62           SEARCHING     STATUTORY       LAW BY COMPUTER (REF-CHYFT71).=
SCRST67                         STEPS           TOWARD A MODEL OF LINGUISTIC PERFORMANCE (REF-PAMFS71).=
MKRS71       M INFORMATION      STORAGE         AND RETRIEVAL SYMPOSIUM .=INTRODUCTION AND PERSPECTIVES FOR THE 1971 AC
KAAEU71      CCESS ARCHIVAL     STORAGE         IN A TIME SHARED ENVIRONMENT.=EFFICIENT UTILIZATION OF LIMITED A
KNGBT71       BALANCED TREE     STORAGE         AND RETRIEVAL ALGORITHM.=A
SEMIS69          INFORMATION    STORAGE         AND RETRIEVAL SYSTEMS (REF-CHYFT71).=
WIAMC70      R HIERARCHICAL     STORAGE         STRUCTURES (REF-WIAFS71).=A METHODOLOGY FOR COMPARISON OF FILE ORGANIZATION AN
SCACS69        COMPUTERIZED     STORAGE         AND RETRIEVAL OF CASE LAW WITHOUT INDEXING: THE HEBREW RESPONSA PROJECT (REF-C
LERRU69      OSITIONAL DATA     STORAGE         AND INFERENCE EXEC (REF-DIRRD71).=RELATIONAL DATA FILE: EXPERIENCE WITH A SYST
REWAR57       RANDOM-ACCESS     STORAGE          (REF-GURER71).=ADDRESSING FOR
FOCIS65          INFORMATION    STORAGE         AND RETRIEVAL USING AVL TREES (REF-KNGBT71).=
GSRRC69       STRUCTURE FOR     STORAGE         AND RETRIEVAL SYSTEMS (REF-GURER71).=A RANDOMIZED COMBINATORIAL FILE
FIJUG70      AN INFORMATION     STORAGE         AND RETRIEVAL SYSTEM ON THE B5500 COMPUTER (REF-HUEHM71).=A USERS GUIDE TO THE
KIGEL69       DIRECT ACCESS     STORAGE          MANIPULATION (REF-HUEHM71).=EXPERIMENTS IN LARGE SCALE COMPUTER
CORIS69          INFORMATION    STORAGE         AND RETRIEVAL (REF-SUSMS71).=
DAOE65       IPLE FILE DATA     STORAGE          SYSTEM : AN APPLICATION OF INTEGER PROGRAMMING (REF-WACOE71).=ON OPTIMAL EXTR
COJEM69      INTENANCE OF A     STORAGE         HIERARCHY FOR AN ON-LINE DATA BASE UNDER TSS/360 (REF-KAAEU71).=ESTABLISHMENT
BEJIS63          INFORMATION    STORAGE         AND RETRIEVAL: TOOLS, ELEMENTS, THEORIES (REF-GURER71).=
BRWGP66      HE PROCESSING,     STORAGE         AND RETRIEVAL OF BIBLIOGRAPHIC INFORMATION AT THE NASA SCIENTIFIC AND TECHNICA
PRFSU39      ND URGENT, THE     STORY           OF CODES AND CIPHERS (REF-MUJDC71) .=SECRET A
LASOS66       AN OUTLINE OF     STRATIFICATIONAL GRAMMAR (REF-PAMFS71).=
HAZSL61                         STRUCTURAL      LINGUISTICS (REF-PAMFS71).=
GURER71      INATORIAL FILE     STRUCTURE       =ELEMENTS OF THE RANDOMIZED COMB
WIAFS71              FILE       STRUCTURE       DETERMINATION.=
SHSNS69            A NET        STRUCTURE       BASED RELATIONAL QUESTION ANSWERER: DESCRIPTION AND EXAMPLES (REF-SUSMS71).=
WEPCR71      ELATIONAL DATA     STRUCTURE        (REF-SUSMS71).=CYLINDERS: A R
```

MALMF69	MODEL FOR FILE	STRUCTURE	DETERMINATION FOR LARGE ON-LINE DATA FILES (REF-WIAFS71).=A
KAMMS70	ND SYSTEM: THE	STRUCTURE	OF THE SEMANTIC FILE (REF-SUSMS71).=THE MI
KAJSS65	THE	STRUCTURE	OF A SEMANTIC THEORY (REF-PAMFS71).=
GSRRC69	INATORIAL FILE	STRUCTURE	FOR STORAGE AND RETRIEVAL SYSTEMS (REF-GURER71).=A RANDOMIZED COMB
FRCSE52	THE	STRUCTURE	OF ENGLISH (REF-DOBHF71).=
HAGGG	ENSE OF PHRASE	STRUCTURE	(REF-DOHHF71).=GENERATIVE GRAMMARS WITHOUT TRANSFORMATIONAL RULES: A DEF
WOWSI67	QUESTIONS ON A	STRUCTURED	DATA BASE (REF-PAMFS71).=SEMANTIC INTERPRETATION OF ENGLISH
UZBOS65	RGANIZATION OF	STRUCTURED	FILES (REF-GURER71).=THE O
WIAMC70	CHICAL STORAGE	STRUCTURES	(REF-WIAFS71).=A METHODOLOGY FOR COMPARISON OF FILE ORGANIZATION AND PROCESSI
LOCMS66	MEMORY	STRUCTURES	IN DEACON NATURAL LANGUAGE QUESTION-ANSWERING SYSTEM (REF-SUSMS71).=
LEUFS69	FILE	STRUCTURES	FOR ON-LINE SYSTEMS (REF-HUEHM71).=
LEUFS69	FILE	STRUCTURES	FOR ON-LINE SYSTEMS (REF-WIAFS71).=
LEUFS67	FILE	STRUCTURES	FOR ON-LINE SYSTEMS (REF-GURER71).=
GLEMP70	ICAL SYNTACTIC	STRUCTURES	(REF-PAMFS71).=THE MULTISTORE PARSER FOR HIERARCH
EVUDS71	DATA	STRUCTURES	AND ALGORITHMIC DETERMINACY (REF-WIAFS71).=
WEJEC66	ROGRAM FOR THE	STUDY	OF NATURAL LANGUAGE COMMUNICATION BETWEEN MAN AND MACHINE (REF-LOUCC71).=ELIZA
KUJAQ67	TER: A LOGICAL	STUDY	(REF-KUJQQ71).=ANSWERING QUESTIONS BY COMPU
MIURD70	AL DATA SYSTEM	STUDY	(REF-MKRS71) .=RELATION
KUJAQ67	TER: A LOGICAL	STUDY	(REF-DIRRD71).=ANSWERING QUESTIONS BY COMPU
GAPPT67	YPING: A PILOT	STUDY	IN SEMANTIC ANALYSIS; (REF-PAMFS71).=PREDICATION I
SIRAE65	BY COMPUTER: A	SURVEY	(REF-MKRS71) (REF-PAMFS71) (REF-SUSMS71) .=ANSWERING ENGLISH QUESTIONS
KAADR67	ER: A CRITICAL	SURVEY	(REF-SUSMS71) (REF-KUJQQ71).=DATA-RETRIEVAL BY COMPUT
COUSG69	A	SURVEY	OF GENERALIZED DATA BASE MANAGEMENT SYSTEMS (REF-MKRS71) .=
REHES47	ELEMENTS OF	SYMBOLIC	LOGIC (REF-KUJQQ71).=
CARIS58	NTRODUCTION TO	SYMBOLIC	LOGIC AND ITS APPLICATIONS (REF-KUJQQ71).=I
FERDS69	DICTIONARY OF	SYMBOLS	OF MATHEMATICAL LOGIC (REF-KUJQQ71).=
MKRS71	AND RETRIEVAL	SYMPOSIUM	.=INTRODUCTION AND PERSPECTIVES FOR THE 1971 ACM INFORMATION STORAGE
DOBHF71	ATURES RESOLVE	SYNTACTIC	AMBIGUITY.=HOW TE
THOPS69	ROGRAM FOR THE	SYNTACTIC	ANALYSIS OF ENGLISH SENTENCES (REF-PAMFS71).=A P
GLEMP70	R HIERARCHICAL	SYNTACTIC	STRUCTURES (REF-PAMFS71).=THE MULTISTORE PARSER FO
KUSAS67	AUTOMATIC	SYNTACTIC	ANALYSIS (REF-PAMFS71).=
KUSMP63	MULTIPLE-PATH	SYNTACTIC	ANALYZER (REF-LOUCC71).=
BOUST68		SYNTACTIC	THEORY IN COMPUTER IMPLEMENTATIONS (REF-PAMFS71).=
SIKIT68	AL THEORIES OF	SYNTAX	(REF-KECCN71).=INTEGRATION OF TRANSFORMATION
SIKIT68	IES ON ENGLISH	SYNTAX	(REF-PAMFS71) .=INTEGRATION OF TRANSFORMATIONAL THEOR
CARLS37	THE LOGICAL	SYNTAX	OF LANGUAGE, (REF-KUJQQ71).=
CHNAT65	THE THEORY OF	SYNTAX	(REF-LOUCC71) (REF-DOBHF71) (REF-PAMFS71) .=ASPECTS OF
LOUCC71	: A CUSTOMIZED	SYSTEM	FOR RESTRICTED, NATURAL ENGLISH.=CUE
SAGSA65	MENT RETRIEVAL	SYSTEM	-- AN ILLUSTRATION (REF-JADOC71).=THE SMART AUTOMATIC DOCU
IEDRS69	A REVIEW OF	SYSTEM	DESIGN TECHNIQUES (REF-WAGQE71).=
VEJED	DECIMAL NUMBER	SYSTEM	(REF-MUJDC71).=ERROR DETECTING AND CORRECTING CODES FOR THE
SAGRA69	AND RETRIEVAL	SYSTEM	EVALUATION (REF-JADOC71).=RELEVANCE ASSESSMENTS
VIHOR65	ON RETRIEVAL	SYSTEM	THEORY (REF-GURER71).=
THFRE69	SIBLE LANGUAGE	SYSTEM	(REF-DOHHF71).=REL: A RAPIDLY EXTEN
KECCN71	ATA MANAGEMENT	SYSTEM	CURRENT STATUS AND PLANS.=THE CONVERSE NATURAL LANGUAGE U
SIISK63	COMMUNICATION	SYSTEM	(REF-HUEHM71) .=SKETCHPAD, A MAN-MACHINE GRAPHICAL
WAUCL69	IN AN ON-LINE	SYSTEM	FOR TEXTUAL ANALYSIS (REF-SUSMS71).=COMPUTATIONAL LINGUISTIC TECHNIQUES
LOFRE69	SIBLE LANGUAGE	SYSTEM	THE REL LANGUAGE PROCESSOR (REF-DOHHF71).=A RAPIDLY EXTEN
LERRD69	ERIENCE WITH A	SYSTEM	FOR PROPOSITIONAL DATA STORAGE AND INFERENCE EXEC (REF-DIRRD71).=RELATIONAL DA
LEMDR68	TION RETRIEVAL	SYSTEM	(REF-KAAEU71) .=DESIGN OF A REVISED ON-LINE INFORMA
LERCS67	A COMPUTER	SYSTEM	FOR INFERENCE EXECUTION AND DATA RETRIEVAL (REF-DIRRD71).=
LOCMS66	TION-ANSWERING	SYSTEM	(REF-SUSMS71).=MEMORY STRUCTURES IN DEACON NATURAL LANGUAGE QUES
LERCS67	A COMPUTER	SYSTEM	FOR INFERENCE EXECUTION AND DATA RETRIEVAL (REF-KUJQQ71).=
KAMMS70	THE MIND	SYSTEM	THE STRUCTURE OF THE SEMANTIC FILE (REF-SUSMS71).=
MIURD70	ELATIONAL DATA	SYSTEM	STUDY (REF-MKRS71) .=R
LOFRE69	SIBLE LANGUAGE	SYSTEM	(REF-SUSMS71).=A RAPIDLY EXTEN
IBMPR67	TURAL LANGUAGE	SYSTEM	(REF-LOUCC71).=PROTO-RELAUES:A RESTRICTIVE NA

IBMTA68 AUTOMATED GRID SYSTEM (TAG): SALES AND SYSTEMS GUIDE (REF-TEDAR71).=THE TIME
FIJGF70 ILE MANAGEMENT SYSTEM (REF-HUEHM71).=A GENERALIZED F
HSUFS70 A FORMAL SYSTEM FOR INFORMATION RETRIEVAL FROM FILES (REF-GURER71).=
FIJUG70 AND RETRIEVAL SYSTEM ON THE B5500 COMPUTER (REF-HUEHM71).=A USERS GUIDE TO THE OPERATION OF AN INFO
HIDLR69 ADER RETRIEVAL SYSTEM (REF-SUSMS71) .=THE LE
DAOE65 A STORAGE SYSTEM : AN APPLICATION OF INTEGER PROGRAMMING (REF-WACQE71).=ON OPTIMAL EXTRACING FR
COKES69 HUMAN BELIEF SYSTEM (REF-SUSMS71) (REF-PAMFS71) .=EXPERIMENTS WITH A SEARCH ALGORITHM FOR THE DAT
COLOL68 TION-ANSWERING SYSTEM WITH NATURAL LANGUAGE AND PICTORIAL INPUT (REF-SUSMS71).=AN ON-LINE QUES
AUSTM64 THE MEDLARS SYSTEM (REF-MKRS71).=
CAJUP65 AR FILE TANDEM SYSTEM (REF-GURER71).=UPLIFTS-UNIVERSITY OF PITTSBURGH LINE
DOGRE69 SIBLE LANGUAGE SYSTEM REL ENGLISH (REF-DOBHF71).=A RAPIDLY EXTEN
BODQA64 TION-ANSWERING SYSTEM FOR HIGH SCHOOL ALGEBRA WORK PROBLEMS (REF-KUJQQ71) (REF-SUSMS71) (REF-LODCC71
EISP68 MANAGEMENT SYSTEM PHASE 1 (REF-WIAFS71).=SYSTEMS PROGRAMMING DOCUMENTATION, TABLES, REMOTE FILE
DAGI68 ILE MANAGEMENT SYSTEM PHASE I (REF-WIAFS71).=GENERAL INFORMATION, REMOTE F
COLSN65 SYSTEMATIZED NOMENCLATURE OF PATHOLOGY (REF-PAMFS71).=
WACQE71 DEOFFS IN FILE SYSTEMS .=QUANTITATIVE EVALUATION OF DESIGN TRA
TEDMD69 ION PROCESSING SYSTEMS (REF-TEDAR71).=METHODOLOGY FOR THE DESIGN OF INFORMAT
SIMNL70 TION-ANSWERING SYSTEMS (REF-SUSMS71) (REF-MKRS71) .=NATURAL LANGUAGE QUES
SIRNL70 TION ANSWERING SYSTEMS (REF-PAMFS71).=NATURAL LANGUAGE QUES
SPJIS65 INFORMATION SYSTEMS SCIENCES: PROCEEDING OF THE SECOND CONGRESS (REF-GURER71).=
SEMIS69 AND RETRIEVAL SYSTEMS (REF-CHYFT71).=INFORMATION STORAGE
KUJQQ71 ATION IN QUERY SYSTEMS =QUANTIFIC
MAJDR67 -TIME COMPUTER SYSTEMS (REF-TEDAR71).=DESIGN OF REAL
NUJDO61 PROCESSING SYSTEMS (REF-WACQE71).=ON THE DESIGN AND OPTIMIZATION OF INFORMATION
LEDFS69 ES FOR ON-LINE SYSTEMS (REF-WIAFS71).=FILE STRUCTUR
LABTA66 OF INFORMATION SYSTEMS (REF-TEDAR71).=THEORETICAL ANALYSIS
MKRGD69 ATA MANAGEMENT SYSTEMS - SOME PRESPECTIVES (REF-MKRS71).=GENERALIZED D
NACAD67 RATELY DEFINED SYSTEMS (REF-TEDAR71).=ACCU
LEDFS67 ES FOR ON-LINE SYSTEMS (REF-GURER71).=FILE STRUCTUR
LASA63 OF INFORMATION SYSTEMS (REF-WACQE71).=SOME APPROACHES TO THE THEORY
MECAI67 OF INFORMATION SYSTEMS (REF-GURER71).=THE ANALYSIS
LEDFS69 ES FOR ON-LINE SYSTEMS (REF-HUEHM71).=FILE STRUCTUR
MCWTA69 ION PROCESSING SYSTEMS (REF-WACQE71).=TOWARD THE AUTOMATIC DESIGN OF DATA ORGANIZATION FOR LARGE SCA
KIPSA69 SYSTEMS ANALYSIS DOCUMENTATION: COMPUTER AIDED DATA DIRECTORY DEFINITION (REF-TEDAR71)
GSRRC69 ND RETRIEVAL SYSTEMS (REF-GURER71).=A RANDOMIZED COMBINATORIAL FILE STRUCTURE FOR STORAGE A
KAJLG68 ORMATION SYSTEMS (REF-MUJDC71).=A LINEAR GEOGRAPHICAL CODE FOR MANAGEMENT INF
IBMTA68 AG): SALES AND SYSTEMS GUIDE (REF-TEDAR71).=THE TIME AUTOMATED GRID SYSTEM (T
GRMSG63 SYSTEMS GENERATION OUTPUT DECOMPOSITION METHOD (REF-TEDAR71).=
GRCSI70 FORMAL CONTROL SYSTEMS (REF-TEDAR71).=SPECIFICATION AND INTERROGATION OF
COUSG69 ASE MANAGEMENT SYSTEMS (REF-MKRS71) .=A SURVEY OF GENERALIZED DATA B
EISP68 SYSTEMS PROGRAMMING DOCUMENTATION, TABLES, REMOTE FILE MANAGEMENT SYSTEM, PHASE 1 (REF
ARJIS69 INFORMATION SYSTEMS IN PERSPECTIVE (REF-GURER71).=
DOGED69 ATA MANAGEMENT SYSTEMS (REF-GURER71).=ELEMENTS OF D
CLCFU66 CE OF INDEXING SYSTEMS (REF-JAUOC71).=FACTORS DETERMINING THE PERFORMAN
EISP68 DOCUMENTATION, TABLES REMOTE FILE MANAGEMENT SYSTEM, PHASE 1 (REF-WIAFS71).=SYSTEMS PROGRAMMING
CAJUP65 GH LINEAR FILE TANDEM SYSTEM (REF-GURER71).=UPLIFTS-UNIVERSITY OF PITTSBUR
DEAPS65 CTURE ORIENTED TARGETING QUERY LANGUAGE (REF-LODCC71).=PHRASE-STRU
QURTL69 THE TEACHABLE LANGUAGE COMPREHENDER (REF-PAMFS71) (REF-SUSMS71) .=
HUECS70 MPUTER SCIENCE TEACHING LABORATORY AT THE UNIVERSITY OF WASHINGTON (REF-HUEHM71).=THE CO
LUHKW69 NTEXT IDEA FOR TECHNICAL LITERATURE (KWIC INDEX). (REF-MKRS71).=KEY-WORD-IN-CO
CHBCT65 COLLOQUIUM ON TECHNICAL PRECONDITIONS FOR RETRIEVAL CENTER OPERATIONS (REF-GURER71).=
BRWGP66 FIC AND TECHNICAL INFORMATION FACILITY (REF-SUSMS71).=GUIDE TO THE PROCESSING, STORAGE AND RETRI
MUJDC71 TA COMPRESSION TECHNIQUES FOR ECONOMIC PROCESSING OF LARGE COMMERCIAL FILES.=DA
WAUCL69 NAL LINGUISTIC TECHNIQUES IN AN ON-LINE SYSTEM FOR TEXTUAL ANALYSIS (REF-SUSMS71).=COMPUTATIO
TEDRS69 SYSTEM DESIGN TECHNIQUES (REF-WACQE71).=A REVIEW OF
NUWCW68 ON WORD CODING TECHNIQUES FOR INFORMATION RETRIEVAL (REF-MUJDC71).=COMPRESSI
CHNCF69 E ORGANIZATION TECHNIQUES (REF-WIAFS71).=A COMPARISON OF FIL

AFCFF67	E ORGANIZATION	TECHNIQUES	(REF-WACQE71).=FORMATTED FIL
PRAIT69	NSFORMATION OF	TERMINAL	MORPHEMES IN MEDICAL ENGLISH (REF-PAMFS71).=IDENTIFICATION AND TRA
CHYFT71	FULL	TEXT	DOCUMENT RETRIEVAL: HEBREW LEGAL TEXTS (REPORT ON THE FIRST PHASE OF THE RESPO
SAGCE68	R INDEXING AND	TEXT	PROCESSING (REF-JADOC71).=COMPUTER EVALUATION FO
SNMMV70	IAD VIRTUES OF	TEXT	COMPACTION (REF-MUJDC71) .=THE MYR
COMMS70	FULL	TEXT	INFORMATION RETRIEVAL (IN HEBREW) (REF-CHYFT71).=
KEATM62		TEXTBOOK	ON MECHANIZED INFORMATION RETRIEVAL (REF-GURER71).=
CHYFT71	: HEBREW LEGAL	TEXTS	(REPORT ON THE FIRST PHASE OF THE RESPONSA RETRIEVAL PROJECT).=FULL TEXT DOCUM
WAUCL69	INE SYSTEM FOR	TEXTUAL	ANALYSIS (REF-SUSMS71).=COMPUTATIONAL LINGUISTIC TECHNIQUES IN AN ON-L
GRCAT69	APPLICATION OF	THEOREM	PROVING TO PROBLEM SOLVING (REF-SUSMS71).=APPLIC
LABTA66		THEORETICAL	ANALYSIS OF INFORMATION SYSTEMS (REF-TEDAR71).=
STRIT68	ANSFORMATIONAL	THEORIES	ON ENGLISH SYNTAX (REF-PAMFS71) .=INTEGRATION OF TR
STRIT68	ANSFORMATIONAL	THEORIES	OF SYNTAX (REF-KECCN71).=INTEGRATION OF TR
BEJIS63	OLS, ELEMENTS,	THEORIES	(REF-GURER71).=INFORMATION STORAGE AND RETRIEVAL: TO
VIBOR65	TRIEVAL SYSTEM	THEORY	(REF-GURER71).=ON RE
SUSST69	A SEMANTIC	THEORY	BASED UPON INTERACTIVE MEANING (REF-SUSMS71).=
WEUES66	NS IN SEMANTIC	THEORY	(REF-PAMFS71).=EXPLORATIO
LASA63	ROACHES TO THE	THEORY	OF INFORMATION SYSTEMS (REF-WACQE71).=SOME APP
HAMAC63	TION OF CODING	THEORY	TO A FILE ADDRESS PROBLEM (REF-GURER71).=AN APPLICA
FEAFI58	OF INFORMATION	THEORY	(REF-KAAEU71).=FOUNDATIONS
FEWIP68	TO PROBABILITY	THEORY	AND ITS APPLICATIONS (REF-GURER71).=AN INTRODUCTION
KAJRI67	ES IN SEMANTIC	THEORY	(REF-PAMFS71).=RECENT ISSU
KAJSS65	OF A SEMANTIC	THEORY	(REF-PAMFS71).=THE STRUCTURE
HAUDT64	DEPENDENCY	THEORY	A FORMALIZATION AND SOME OBSERVATIONS (REF-PAMFS71).=
FICTM66	OWARD A MODERN	THEORY	OF CASE (REF-PAMFS71).=T
BOUST68	SYNTACTIC	THEORY	IN COMPUTER IMPLEMENTATIONS (REF-PAMFS71).=
CHCIT61	INFORMATION	THEORY	(REF-GURER71).=
ASRIT65	INFORMATION	THEORY	(REF-KAAEU71).=
AQLNA65	TO THE LOGICAL	THEORY	OF INTERROGATIVES, PART I, ANALYSIS (REF-KUJQQ71).=A NEW APPROACH
CHNAT65	ASPECTS OF THE	THEORY	OF SYNTAX (REF-LODCC71) (REF-DOBHF71) (REF-PAMFS71) .=ASPECT
BUVAW45	AS WE MAY	THINK	(REF-MKRS71).=
KAAEU71	L STORAGE IN A	TIME	SHARED ENVIRONMENT.=EFFICIENT UTILIZATION OF LIMITED ACCESS ARCHIVA
IBMTA68	THE	TIME	AUTOMATED GRID SYSTEM (TAG): SALES AND SYSTEMS GUIDE (REF-TEDAR71).=
LERTM65	L DATA FILE: A	TOOL	FOR MECHANIZED INFERENCE EXECUTION AND DATA RETRIEVAL (REF-KECCN71).=RELATIONA
LERRU65	L DATA FILE: A	TOOL	FOR MECHANIZED INFERENCE EXECUTION AND DATA RETRIEVAL (REF-KUJQQ71).=RELATIONA
ELW CT63	COMPUTER AS A	TOOL	FOR LEGAL RESEARCH (REF-CHYFT71).=THE
LEUMT64	THE MACHINE	TOOLS	OF COMBINATORICS (REF-GURER71).=
FRAC169	L AS AUXILIARY	TOOLS	IN HEBREW LEGAL RESEARCH (IN HEBREW) (REF-CHYFT71).=THE COMPUTER AND INFORMA
BEJIS63	AND RETRIEVAL:	TOOLS	ELEMENTS, THEORIES (REF-GURER71).=INFORMATION STORAGE
WACQE71	TION OF DESIGN	TRADEOFFS	IN FILE SYSTEMS .=QUANTITATIVE EVALUA
ASWTR68		TRAMP	AN INTERPRETIVE ASSOCIATIVE PROCESSOR WITH DEDUCTIVE CAPABILITIES (REF-KECCN7
OVCCI65	ON INFORMATION	TRANSFER	EXPERIMENTS (INTREX) (REF-MKRS71) .=PLANNING CONFERENCE
SCGMK63	KEY-TO-ADDRESS	TRANSFORMATION	(REF-GURER71).=A METHOD FOR
PRAIT69	TIFICATION AND	TRANSFORMATION	OF TERMINAL MORPHEMES IN MEDICAL ENGLISH (REF-PAMFS71).=IDEN
STRIT68	INTEGRATION OF	TRANSFORMATIONAL	THEORIES OF SYNTAX (REF-KECCN71).=INTEGR
STRIT68	INTEGRATION OF	TRANSFORMATIONAL	THEORIES ON ENGLISH SYNTAX (REF-PAMFS71) .=INTEGR
HAGGG	AMMARS WITHOUT	TRANSFORMATIONAL	RULES: A DEFENSE OF PHRASE STRUCTURE (REF-DOBHF71).=GENERATIVE GR
WOWTN70		TRANSITION	NETWORK GRAMMARS FOR NATURAL LANGUAGE ANALYSIS (REF-KECCN71) (REF-DOBHF71) .=
CHWTK62		TRANSLATION	OF RETRIEVAL REQUESTS COUCHED IN A 'SEMIFORMAL' ENGLISH-LIKE LANGUAGE, (REF-MK
CESMT	MECHANICAL	TRANSLATION	THE CORRELATIONAL SOLUTION (REF-PAMFS71).=
FARTI61		TRANSMISSION	OF INFORMATION (REF-KAAEU71).=
KEWET68	ERRORS IN	TRANSMISSION	OF COMPRESSED DATA (REF-MUJDC71).=
KNGBT71	A BALANCED	TREE	STORAGE AND RETRIEVAL ALGORITHM.=

FEDBT68	BALANCED	TREE	SEARCHING (REF-KNGBT71).=
WIPTF60		TREES	FORESTS, AND REARRANGING (REF-KNGBT71).=
HAJMT67	TY MATRICES BY	TREES	(REF-JADOC71).=REPRESENTATION OF SIMILARI
HITSC62	IES OF CERTAIN	TREES	(REF-KNGBT71).=SOME COMBINATORIAL PROPERT
KNUOB70	BINARY SEARCH	TREES	(REF-KNGBT71).=OPTIMUM
FOCIS65	EVAL USING AVL	TREES	(REF-KNGBT71).=INFORMATION STORAGE AND RETRI
YONIS69	VELOPMENT OF A	TRUE	SCIENTIFIC DISCIPLINE (REF-SUSMS71).=INFORMATION SCIENCE: TOWARD THE DE
RUBIM62	TO MEANING AND	TRUTH	(REF-KUJQQ71).=AN INQUIRY IN
IBMDM70	FACILITIES OF	TSS/360	(REF-KAAEU71).=DATA MANAGEMENT
COJEM69	ATA BASE UNDER	TSS/360	(REF-KAAEU71).=ESTABLISHMENT AND MAINTENANCE OF A STORAGE HIERARCHY FOR AN ON
GAPPT67	PREDICATION	TYPING	A PILOT STUDY IN SEMANTIC ANALYSIS; (REF-PAMFS71).=
COJEM69	LINE DATA BASE	UNDER	TSS/360 (REF-KAAEU71).=ESTABLISHMENT AND MAINTENANCE OF A STORAGE HIERARCHY FO
SIRCM68	ODEL OF VERBAL	UNDERSTANDING	(REF-PAMFS71).=A COMPUTATIONAL M
SIRCM68	ODEL OF VERBAL	UNDERSTANDING	(REF-SUSMS71).=A COMPUTATIONAL M
MOJTU67	TOWARDS THE	UNDERSTANDING	OF NATURAL LANGUAGES BY MACHINES (REF-LODCC71).=
HOCPU66	THE PROBLEM OF	UNIVERSALS	IN LANGUAGE (REF-PAMFS71).=THE PR
HUECS70	ORATORY AT THE	UNIVERSITY	OF WASHINGTON (REF-HUEHM71).=THE COMPUTER SCIENCE TEACHING LAB
DIRRU69	THE RECURSIVE	UNSOLVABILITY	OF THE DECISION PROBLEM FOR THE CLASS OF DEFINITE FORMULAS (REF-DIRRU71).=
CAJUP65		UPLIFTS-UNIVERSITY	OF PITTSBURGH LINEAR FILE TANDEM SYSTEM (REF-GURER71).=
SUSST69	C THEORY BASED	UPON	INTERACTIVE MEANING (REF-SUSMS71).=A SEMANTI
PRESU39	SECRET AND	URGENT	THE STORY OF CODES AND CIPHERS (REF-MUJDC71) .=
MOJPR69	REPORT ON THE	USE	OF ENGLISH IN INFORMATION RETRIEVAL (REF-PAMFS71) .=A PROGRESS
RONUE68	THE	USE	OF ENGLISH FOR COMMUNICATING WITH A COMPUTER (REF-LODCC711.=
FIJUG70	A	USERS	GUIDE TO THE OPERATION OF AN INFORMATION STORAGE AND RETRIEVAL SYSTEM ON THE B
IVKAU68	APL/360:	USERS	MANUAL (REF-HUEHM71).=
MAWFC68	LE COMPRESSION	USING	HUFFMAN CODING (REF-MUJDC71).=FI
FOCIS65	AND RETRIEVAL	USING	AVL TREES (REF-KNGBT71).=INFORMATION STORAGE
KAAEU71	EFFICIENT	UTILIZATION	OF LIMITED ACCESS ARCHIVAL STORAGE IN A TIME SHARED ENVIRONMENT.=
QUWVE60		VARIABLES	EXPLAINED AWAY (REF-KECCN71).=
NEPEE61	ERROR-LIMITING	VARIABLE-LENGTH	CODES (REF-MUJDC71).=EFFICIENT
CHYAG66	OF THE HEBREW	VERB	(IN HEBREW) (REF-CHYFT71).=AUTOMATIC GRAMMATICAL ANALYSIS
CHYAG67	OF THE HEBREW	VERB	(IN HEBREW) ENGLISH ABSTRACT (REF-CHYFT71).=AUTOMATIC GRAMMATICAL ANALYSIS
SIRCM68	IONAL MODEL OF	VERBAL	UNDERSTANDING (REF-PAMFS71).=A COMPUTAT
SIRCM68	IONAL MODEL OF	VERBAL	UNDERSTANDING (REF-SUSMS71).=A COMPUTAT
SCG67	AL: A CRITICAL	VIEW	(REF-GURER71).=INFORMATION RETRIEV
GAPLV63	A LINGUISTICS	VIEW	OF LANGUAGE DATA PROCESSING (REF-PAMFS71).=
SNMMV70	THE MYRIAD	VIRTUES	OF TEXT COMPACTION (REF-MUJDC71) .=
HUECS70	UNIVERSITY OF	WASHINGTON	(REF-HUEHM71).=THE COMPUTER SCIENCE TEACHING LABORATORY AT THE
NUWCW68	COMPRESSION	WORD	CODING TECHNIQUES FOR INFORMATION RETRIEVAL (REF-MUJDC71).=
CHECS69	ON OF CALCULUS	WORD	PROBLEMS (REF-SUSMS71).=COMPUTER SOLUTI
EALAD67	EED OF ENGLISH	WORDS	(REF-PAMFS71).=AUTOMATIC DETERMINATION OF PARTS OF SP
BOUQA64	SCHOOL ALGEBRA	WORK	PROBLEMS (REF-KUJQQ71) (REF-SUSMS71) (REF-LODCC71).=A QUESTION-ANSWERING SYSI

OVCCI65	XPERIMENTS	(INTREX)	(REF-MKRS71) .=PLANNING CONFERENCE ON INFORMATION TRANSFER E
LUHKW59	CAL LITERATURE	(KWIC	INDEX). (REF-MKRS71).=KEY-WORD-IN-CONTEXT IDEA FOR TECHNI
WIRCR62		(REF-CHYFT71)	=COMPUTER RETRIEVAL OF CASE LAW
SLMOR70		(REF-CHYFT71)	=THE OCCULT IN THE RIVASH RESPONSA (IN HEBREW)
SLMMS69		(REF-CHYFT71)	=MARRANOS SEARCH IN THE RIVASH RESPONSA (IN HEBREW)
SCACS69	PROJECT	(REF-CHYFT71)	=COMPUTERIZED STORAGE AND RETRIEVAL OF CASE LAW WITHOUT INDEXING: THE HEBREW R
SEMIS69	VAL SYSTEMS	(REF-CHYFT71)	=INFORMATION STORAGE AND RETRIE
NIGSP69		(REF-CHYFT71)	=THE STATUS PROJECT:SEARCHING ATOMIC ENERGY LAW BY COMPUTER
KAMLP62		(REF-CHYFT71)	=A LONG-RANGE PROGRAM FOR MECHANIZED LEGAL AND PATENT SEARCHING CENTERS
HOJSS62	PUTER	(REF-CHYFT71)	=SEARCHING STATUTORY LAW BY COM
CHYAG66		(REF-CHYFT71)	=AUTOMATIC GRAMMATICAL ANALYSIS OF THE HEBREW VERB (IN HEBREW)
FRAMR68		(REF-CHYFT71)	=MODERN RETRIEVAL METHODS APPLIED TO JEWISH CASE LAW
COMMS70		(REF-CHYFT71)	=FULL TEXT INFORMATION RETRIEVAL (IN HEBREW)
CHYAG67		(REF-CHYFT71)	=AUTOMATIC GRAMMATICAL ANALYSIS OF THE HEBREW VERB (IN HEBREW) ENGLISH ABSTRA
FRALI68		(REF-CHYFT71)	=LEGAL INFORMATION RETRIEVAL
FRAC169	BREW)	(REF-CHYFT71)	=THE COMPUTER AND INFORMATION RETRIEVAL AS AUXILIARY TOOLS IN HEBREW LEGAL RES
ELW CT63		(REF-CHYFT71)	=THE COMPUTER AS A TOOL FOR LEGAL RESEARCH
BEJOA69		(REF-CHYFT71)	=ON-LINE ACCESS TO INFORMATION: NSF AS AN AID TO THE INDEXER/CATALOGER
SHJML67		(REF-DIRRD71)	=MATHEMATICAL LOGIC
MEEIM64	OGIC	(REF-DIRRD71)	=INTRODUCTION TO MATHEMATICAL L
MAMRD66	PHILOSOPHY	(REF-DIRRD71)	=RELATIONAL DATA FILE I: DESIGN
LERCS67		(REF-DIRRD71)	=A COMPUTER SYSTEM FOR INFERENCE EXECUTION AND DATA RETRIEVAL
LERRD66	MENTATION	(REF-DIRRD71)	=RELATIONAL DATA FILE II: IMPLE
LERRD69		(REF-DIRRD71)	=RELATIONAL DATA FILE: EXPERIENCE WITH A SYSTEM FOR PROPOSITIONAL DATA STORAGE
KUJIR70		(REF-DIRRD71)	=INTERROGATING A RELATIONAL DATA FILE: REMARKS ON THE ADMISSIBILITY OF INPUT Q
KUJAQ67		(REF-DIRRD71)	=ANSWERING QUESTIONS BY COMPUTER: A LOGICAL STUDY
ACWSC54	PROBLEM	(REF-DIRRD71)	=SOLVABLE CASES OF THE DECISION
DIRRU69		(REF-DIRRD71)	=THE RECURSIVE UNSOLVABILITY OF THE DECISION PROBLEM FOR THE CLASS OF DEFINITE
THFRE69		(REF-DOBHF71)	=REL: A RAPIDLY EXTENSIBLE LANGUAGE SYSTEM
WOWTN70	F-KECCN71)	(REF-DOBHF71)	.=TRANSITION NETWORK GRAMMARS FOR NATURAL LANGUAGE ANALYSIS (RE
LOPRE69		(REF-DOBHF71)	=A RAPIDLY EXTENSIBLE LANGUAGE SYSTEM: THE REL LANGUAGE PROCESSOR
KAHIC69		(REF-DOBHF71)	=(CIRCULATED LETTER)
KAMEP67		(REF-DOBHF71)	(REF-KECCN71).=EXPERIMENTS WITH A POWERFUL PARSER
HAGGG		(REF-DOBHF71)	=GENERATIVE GRAMMARS WITHOUT TRANSFORMATIONAL RULES: A DEFENSE OF PHRASE STRUC
FRCSE52		(REF-DOBHF71)	=THE STRUCTURE OF ENGLISH
CHMAT65	(REF-LOUCC71)	(REF-DOBHF71)	(REF-PAMFS71) .=ASPECTS OF THE THEORY OF SYNTAX
EAJGR65		(REF-DOBHF71)	=GENERATING A RECOGNIZER FOR A BNF GRAMMAR
DOJ		(REF-DOBHF71)	=DEVOTIONS
DOGRE69		(REF-DOBHF71)	=A RAPIDLY EXTENSIBLE LANGUAGE SYSTEM: REL ENGLISH
WACIR65		(REF-GURER71)	=INFORMATION RETRIEVAL
SCG67	CAL VIEW	(REF-GURER71)	=INFORMATION RETRIEVAL: A CRITI
SAGAI63	(REF-MKRS71)	(REF-GURER71)	=AUTOMATIC INFORMATION ORGANIZATION AND RETRIEVAL
TAMNU66		(REF-GURER71)	=NON-UNIFORM KEY DISTRIBUTION AND ADDRESS CALCULATION SORTING
TOJAA66		(REF-GURER71)	=ANALYSIS OF THE ADDRESS ASSIGNMENT PROBLEM FOR CLUSTERED KEYS
VIBOR65		(REF-GURER71)	=ON RETRIEVAL SYSTEM THEORY
SPJIS65		(REF-GURER71)	=INFORMATION SYSTEMS SCIENCES: PROCEEDING OF THE SECOND CONGRESS

SAGDM66		(REF-GURER71)	=DATA MANIPULATION AND PROGRAMMING PROBLEMS IN AUTOMATIC INFORMATION RETRIEVAL
SCGMK63		(REF-GURER71)	=A METHOD FOR KEY-TO-ADDRESS TRANSFORMATION
LEDMT64	RICS	(REF-GURER71)	=THE MACHINE TOOLS OF COMBINATO
MECMM70		(REF-GURER71)	=MAN-MACHINE COMMUNICATIONS
PEWAR57	TORAGE	(REF-GURER71)	=ADDRESSING FOR RANDOM-ACCESS S
MECAI67	STEMS	(REF-GURER71)	=THE ANALYSIS OF INFORMATION SY
LEDFS67	STEMS	(REF-GURER71)	=FILE STRUCTURES FOR ON-LINE SY
LUVMR70		(REF-GURER71)	=MULTI-ATTRIBUTE RETRIEVAL WITH COMBINED INDEXES
GAPNL63	TER	(REF-GURER71)	=NATURAL LANGUAGE AND THE COMPU
KEAEI65	G	(REF-GURER71)	=ELECTRONIC INFORMATION HANDLIN
GULEI63	ACHINES	(REF-GURER71)	=ELECTRONIC INFORMATION-LOGIC M
HAMAC63		(REF-GURER71)	=AN APPLICATION OF CODING THEORY TO A FILE ADDRESS PROBLEM
KEATM62		(REF-GURER71)	=TEXTBOOK ON MECHANIZED INFORMATION RETRIEVAL
GSRRC69		(REF-GURER71)	=A RANDOMIZED COMBINATORIAL FILE STRUCTURE FOR STORAGE AND RETRIEVAL SYSTEMS
KIJMA61		(REF-GURER71)	=A MAGNETIC ASSOCIATIVE MEMORY
HADIC67		(REF-GURER71)	=INTRODUCTION TO COMPUTATIONAL LINGUISTICS
GHSAF68	TTRIBUTES	(REF-GURER71)	=APPLICATION OF FINITE GEOMETRY IN FILE ORGANIZATION FOR RECORDS WITH MULTIPLE
FEWIP68		(REF-GURER71)	=AN INTRODUCTION TO PROBABILITY THEORY AND ITS APPLICATIONS
GOSBP65		(REF-GURER71)	=BACKTRACK PROGRAMMING
HSUFS70		(REF-GURER71)	=A FORMAL SYSTEM FOR INFORMATION RETRIEVAL FROM FILES
KOAOD66		(REF-GURER71)	=AN OCCUPANCY DISCIPLINE AND APPLICATIONS,
CAJUP65		(REF-GURER71)	=UPLIFTS-UNIVERSITY OF PITTSBURGH LINEAR FILE TANDEM SYSTEM
ARJIS69	TIVE	(REF-GURER71)	=INFORMATION SYSTEMS IN PERSPEC
DOGED69	STEMS	(REF-GURER71)	=ELEMENTS OF DATA MANAGEMENT SY
UZBOS65	FILES	(REF-GURER71)	=THE ORGANIZATION OF STRUCTURED
CHBSH63		(REF-GURER71)	=SHIFF: A REALIZABLE FORM OF ASSOCIATIVE MEMORY
CHBCT65		(REF-GURER71)	=COLLOQUIUM ON TECHNICAL PRECONDITIONS FOR RETRIEVAL CENTER OPERATIONS
BEJIS63		(REF-GURER71)	=INFORMATION STORAGE AND RETRIEVAL: TOOLS, ELEMENTS, THEORIES
CHCIT61		(REF-GURER71)	=INFORMATION THEORY
BOCMI63	G	(REF-GURER71)	=METHODS OF INFORMATION HANDLIN
LEDFS69	STEMS	(REF-HUEHM71)	=FILE STRUCTURES FOR ON-LINE SY
STISK63		(REF-HUEHM71)	.=SKETCHPAD, A MAN-MACHINE GRAPHICAL COMMUNICATION SYSTEM
IVKAU68		(REF-HUEHM71)	=APL/360: USERS MANUAL
KIGEL69		(REF-HUEHM71)	=EXPERIMENTS IN LARGE SCALE COMPUTER DIRECT ACCESS STORAGE MANIPULATION
KIGAL70		(REF-HUEHM71)	=APL/B5500: THE LANGUAGE AND ITS IMPLEMENTATION
KNRAC69		(REF-HUEHM71)	=THE ART OF COMPUTER PROGRAMMING VOL. 1
HUECS70		(REF-HUEHM71)	=THE COMPUTER SCIENCE TEACHING LABORATORY AT THE UNIVERSITY OF WASHINGTON
FIJGF70	SYSTEM	(REF-HUEHM71)	=A GENERALIZED FILE MANAGEMENT
FIJUG70	COMPUTER	(REF-HUEHM71)	=A USERS GUIDE TO THE OPERATION OF AN INFORMATION STORAGE AND RETRIEVAL SYSTEM
SAGRA69		(REF-JADOC71)	=RELEVANCE ASSESSMENTS AND RETRIEVAL SYSTEM EVALUATION
SAGSA65		(REF-JADOC71)	=THE SMART AUTOMATIC DOCUMENT RETRIEVAL SYSTEM -- AN ILLUSTRATION
TANP65		(REF-JADOC71)	=A NOTE ON THE PSEUDO-MATHEMATICS OF RELEVANCE
SPKCA67		(REF-JADOC71)	=CURRENT APPROACHES TO CLASSIFICATION AND CLUMP-FINDING AT THE CLRU
SAGCE68		(REF-JADOC71)	=COMPUTER EVALUATION FOR INDEXING AND TEXT PROCESSING
SWJEI67		(REF-JADOC71)	=EFFECTIVENESS OF INFORMATION RETRIEVAL METHODS
REACH64		(REF-JADOC71)	=THE CONSISTENCY OF HUMAN JUDGEMENTS OF RELEVANCE
NAGSA68		(REF-JADOC71)	=STATE OF THE ART IN PATTERN RECOGNITION
JADBI69		(REF-JADOC71)	=BASIS FOR AN IMPROVABILITY MEASURE IN INFORMATION RETRIEVAL
HAJMT67		(REF-JADOC71)	=REPRESENTATION OF SIMILARITY MATRICES BY TREES
JADCR		(REF-JADOC71)	=CLASSIFICATION, RELEVANCE AND INFORMATION RETRIEVAL
JADNS69		(REF-JADOC71)	=A NOTE ON A SET OF FUNCTIONS FOR INFORMATION RETRIEVAL
EVTHP64		(REF-JADOC71)	=A HEURISTIC PROGRAM TO SOLVE GEOMETRIC-ANALOGY PROBLEMS
DOLSR		(REF-JADOC71)	=SEMANTIC ROAD MAPS FOR LITERATURE SEARCHERS
DOLIA62	SOCIATION	(REF-JADOC71)	=INDEXING AND ABSTRACTING BY AS
CLCFD66		(REF-JADOC71)	=FACTORS DETERMINING THE PERFORMANCE OF INDEXING SYSTEMS
LEMDR68		(REF-KAAEU71)	.=DESIGN OF A REVISED ON-LINE INFORMATION RETRIEVAL SYSTEM
IBMDM70	TSS/360	(REF-KAAEU71)	=DATA MANAGEMENT FACILITIES OF

```
HUDMC                          (REF-KAAEU71)     =A METHOD FOR THE CONSTRUCTION OF MINIMUM REDUNDANCY CODES
INTRX70                        (REF-KAAEU71)     .=PROJECT INTREX SEMIANNUAL ACTIVITY REPORT
FARTI61                        (REF-KAAEU71)     =TRANSMISSION OF INFORMATION.
FEAFI58      ORY               (REF-KAAEU71)     =FOUNDATIONS OF INFORMATION THE
ASKIT65                        (REF-KAAEU71)     =INFORMATION THEORY
COJEM69      U                 (REF-KAAEU71)     =ESTABLISHMENT AND MAINTENANCE OF A STORAGE HIERARCHY FOR AN ON-LINE DATA BASE
WOWTN70                        (REF-KECCN71)      (REF-DOBHF71) .=TRANSITION NETWORK GRAMMARS FOR NATURAL LANGUAGE ANALYSIS
STRIT68                        (REF-KECCN71)     =INTEGRATION OF TRANSFORMATIONAL THEORIES OF SYNTAX
GURSM68      IC MEMORY         (REF-KECCN71)      (REF-PAMES71) .=SEMANT
LERTM65                        (REF-KECCN71)     =RELATIONAL DATA FILE: A TOOL FOR MECHANIZED INFERENCE EXECUTION AND DATA RETR
QUWVE60                        (REF-KECCN71)     =VARIABLES EXPLAINED AWAY
KAJPL66                        (REF-KECCN71)     =THE PHILOSOPHY OF LANGUAGE
KECDM68                        (REF-KECCN71)     =DATA MANAGEMENT IN ORDINARY ENGLISH: EXAMPLES
KECNL68                        (REF-KECCN71)     =A NATURAL LANGUAGE COMPILER FOR ONLINE DATA MANAGEMENT
KAMEP67       (REF-DOBHF71)    (REF-KECCN71)     =EXPERIMENTS WITH A POWERFUL PARSER
ASWTR68                        (REF-KECCN71)     =TRAMP: AN INTERPRETIVE ASSOCIATIVE PROCESSOR WITH DEDUCTIVE CAPABILITIES
CARMN56      (REF-KUJGQ71)     (REF-KECCN71)      .=MEANING AND NECESSITY
FEJAB69      SSOR              (REF-KECCN71)     =AN ALGOL-BASED ASSOCIATE PROCE
WIPTF60      G                 (REF-KNGBT71)     =TREES, FORESTS, AND REARRANGIN
KNUAC68                        (REF-KNGBT71)     =THE ART OF COMPUTER PROGRAMMING, VOL 1: FUNDAMENTAL ALGORITHMS
GHSF069       (REF-WACQE71)    (REF-KNGBT71)     =FILE ORGANIZATION: ON THE SELECTION OF RANDOM ACCESS INDEX POINTS FOR SEQUENT
HITSC62                        (REF-KNGBT71)     =SOME COMBINATORIAL PROPERTIES OF CERTAIN TREES
KNUOB70                        (REF-KNGBT71)     =OPTIMUM BINARY SEARCH TREES
BOAEN60                        (REF-KNGBT71)     =ON THE EFFICIENCY OF A NEW METHOD OF DICTIONARY CONSTRUCTION
FOCIS65                        (REF-KNGBT71)     =INFORMATION STORAGE AND RETRIEVAL USING AVL TREES
ADGA0                          (REF-KNGBT71)     =AN ALGORITHM FOR THE ORGANIZATION OF INFORMATION
FEUBT68                        (REF-KNGBT71)     =BALANCED TREE SEARCHING
WHAPM50      EDITION)          (REF-KUJGQ71)     =PRINCIPIA MATHEMATICA (SECOND
TAACR41                        (REF-KUJGQ71)     =ON THE CALCULUS OF RELATIONS
RUBIM62      UTH               (REF-KUJGQ71)     =AN INQUIRY INTO MEANING AND TR
LERCS67                        (REF-KUJGQ71)     =A COMPUTER SYSTEM FOR INFERENCE EXECUTION AND DATA RETRIEVAL
REMES47                        (REF-KUJGQ71)     =ELEMENTS OF SYMBOLIC LOGIC
LERRD65                        (REF-KUJGQ71)     =RELATIONAL DATA FILE: A TOOL FOR MECHANIZED INFERENCE EXECUTION AND DATA RETR
KUJAQ67                        (REF-KUJGQ71)     =ANSWERING QUESTIONS BY COMPUTER: A LOGICAL STUDY
KUJLA69                        (REF-KUJGQ71)     =LOGICAL ASPECTS OF QUESTION-ANSWERING BY COMPUTER
HADCL63                        (REF-KUJGQ71)     =COMMUNICATION: A LOGICAL MODEL
JEOPG65                        (REF-KUJGQ71)     =THE PHILOSOPHY OF GRAMMAR
KUJIR70                        (REF-KUJGQ71)     =INTERROGATING A RELATIONAL DATA FILE: REMARKS ON THE ADMISSIBILITY OF INPUT Q
KAADR67      REF-SUSMS71)      (REF-KUJGQ71)     =DATA-RETRIEVAL BY COMPUTER: A CRITICAL SURVEY (
AQLNA65                        (REF-KUJGQ71)     =A NEW APPROACH TO THE LOGICAL THEORY OF INTERROGATIVES, PART I, ANALYSIS
CARMN56      ND NECESSITY      (REF-KUJGQ71)      (REF-KECCN71) .=MEANING A
BENAQ63                        (REF-KUJGQ71)     =AN ANALYSIS OF QUESTIONS: PRELIMINARY REPORT
BOUQA64       PROBLEMS         (REF-KUJGQ71)     (REF-SUSMS71) (REF-LODCC71).=A QUESTION-ANSWERING SYSTEM FOR HIGH SCHOOL ALGEB
CARLS57                        (REF-KUJGQ71)     =THE LOGICAL SYNTAX OF LANGUAGE,
CARIS58                        (REF-KUJGQ71)     =INTRODUCTION TO SYMBOLIC LOGIC AND ITS APPLICATIONS
FERDS69                        (REF-KUJGQ71)     =DICTIONARY OF SYMBOLS OF MATHEMATICAL LOGIC
WEJEC66      MACHINE           (REF-LODCC71)     =ELIZA:A COMPUTER PROGRAM FOR THE STUDY OF NATURAL LANGUAGE COMMUNICATION BETW
MITEP64      NUAL              (REF-LODCC71)     =MITRE, ENGLISH PREPROCESSOR MA
MOJTU67                        (REF-LODCC71)     =TOWARDS THE UNDERSTANDING OF NATURAL LANGUAGES BY MACHINES
LOUIE68                        (REF-LODCC71)     =AN INTRODUCTION TO ELF - LIST PROCESSING IN PL/I
RONUE68                        (REF-LODCC71)     =THE USE OF ENGLISH FOR COMMUNICATING WITH A COMPUTER
ROPIC66      H                 (REF-LODCC71)     =THE IBM CORF GRAMMAR OF ENGLIS
IBMPR67                        (REF-LODCC71)     =PROTO-RELADES:A RESTRICTIVE NATURAL LANGUAGE SYSTEM
KUSMP63      ER                (REF-LODCC71)     =MULTIPLE-PATH SYNTACTIC ANALYZ
DEAPS65                        (REF-LODCC71)     =PHRASE-STRUCTURE ORIENTED TARGETING QUERY LANGUAGE
BOUQA64       (REF-SUSMS71)    (REF-LODCC71)     =A QUESTION-ANSWERING SYSTEM FOR HIGH SCHOOL ALGEBRA WORK PROBLEMS (REF-KUJGQ7
CHNAT65                        (REF-LODCC71)     (REF-DOBHF71) (REF-PAMES71) .=ASPECTS OF THE THEORY OF SYNTAX
```

```
CUPAR69                    (REF-LODCC71)     =AN AUTOMATED RECOGNTION GRAMMAR FOR ENGLISH
CHTGI68      MAR           (REF-LODCC71)     =ON THE GENERATION AND IMPLEMENTATION OF REDUCTIONS ANALYSIS PROGRAMS FOR CONT
STMAI70                    (REF-MKRS71)      =AUTOMATIC INDEXING: A STATE OF THE ART REPORT
SAGAI63                    (REF-MKRS71)       (REF-GURER71).=AUTOMATIC INFORMATION ORGANIZATION AND RETRIEVAL
YNCMI62                    (REF-MKRS71)      =COMIT AS AN IR LANGUAGE
SIRAE65                    (REF-MKRS71)       (REF-PAMFS71) (REF-SUSMS71) .=ANSWERING ENGLISH QUESTIONS BY COMPUTER: A SURV
SIMNL70    REF-SUSMS71)    (REF-MKRS71)      .=NATURAL LANGUAGE QUESTION-ANSWERING SYSTEMS. (
LUHKW59                    (REF-MKRS71)      =KEY-WORD-IN-CONTEXT IDEA FOR TECHNICAL LITERATURE (KWIC INDEX).
MIJRU70                    (REF-MKRS71)      .=RELATIONAL DATA SYSTEM STUDY
MKRGU69                    (REF-MKRS71)      =GENERALIZED DATA MANAGEMENT SYSTEMS - SOME PRESPECTIVES
SGHSA64                    (REF-MKRS71)      .=STATISTICAL ASSOCIATION METHODS FOR MACHANIZED DOCUMENTATION.
OVCCI65                    (REF-MKRS71)      .=PLANNING CONFERENCE ON INFORMATION TRANSFER EXPERIMENTS (INTREX)
KIGLC63    ONGRESS         (REF-MKRS71)      =AUTOMATION AND THE LIBRARY OF C
GWCLB61                    (REF-MKRS71)      =BASEBALL: AN AUTOMATIC QUESTION-ANSWERER.
AUSTM64                    (REF-MKRS71)      =THE MEDLARS SYSTEM.
AFICP69                    (REF-MKRS71)      =AFIPS 1969 SPRING JOINT COMPUTER CONFERENCE
CGMIE58                    (REF-MKRS71)      =INFORMATION CONTROL
COUSG69                    (REF-MKRS71)      .=A SURVEY OF GENERALIZED DATA BASE MANAGEMENT SYSTEMS
CHWTR62                    (REF-MKRS71)      =TRANSLATION OF RETRIEVAL REQUESTS COUCHED IN A 'SEMIFORMAL' ENGLISH-LIKE LANG
BUVAW45                    (REF-MKRS71)      =AS WE MAY THINK
SCEDM63                    (REF-MUJDC71)     =A DICTIONARY FOR MINIMAL REDUNDANCY ENCODING
WRMML60                    (REF-MUJDC71)     =MECHANIZING A LARGE INDEX; APPENDIX: THE SOUNDEX CODE
SNMMV70    MPACTION        (REF-MUJDC71)     .=THE MYRIAD VIRTUES OF TEXT CO
SCELE67                    (REF-MUJDC71)     =A LANGUAGE ELEMENT FOR COMPRESSION CODING
VEJED                      (REF-MUJDC71)     =ERROR DETECTING AND CORRECTING CODES FOR THE DECIMAL NUMBER SYSTEM
NUWAW                      (REF-MUJDC71)     =RADC-TDR-62-13, VOL. 1
OVCIN70                    (REF-MUJDC71)     =PROJECT INTREX SEMIANNUAL ACTIVITY REPORT
MABAD67                    (REF-MUJDC71)     =AUTOMATIC DATA COMPRESSION
PRFSU39                    (REF-MUJDC71)     .=SECRET AND URGENT, THE STORY OF CODES AND CIPHERS
NEPEE61                    (REF-MUJDC71)     =EFFICIENT ERROR-LIMITING VARIABLE-LENGTH CODES
NUWCW68                    (REF-MUJDC71)     =COMPRESSION WORD CODING TECHNIQUES FOR INFORMATION RETRIEVAL
MAWFC68     CODING         (REF-MUJDC71)     =FILE COMPRESSION USING HUFFMAN
LOEDC68                    (REF-MUJDC71)     =DATA CODE FOR CALENDAR DATE FOR MACHINE-TO-MACHINE DATA INTERCHANGE
KAUTC67                    (REF-MUJDC71)     =THE CODEBREAKERS
KEWET68                    (REF-MUJDC71)     =ERRORS IN TRANSMISSION OF COMPRESSED DATA
KULCG63                    (REF-MUJDC71)     =CODING OF GRAMMATICAL DATA
GAHCR56                    (REF-MUJDC71)     =CRYPTANALYSIS
KAJLG68                    (REF-MUJDC71)     =A LINEAR GEOGRAPHICAL CODE FOR MANAGEMENT INFORMATION SYSTEMS
KECPA70    ANGE            (REF-MUJDC71)     =PROPOSED AMERICAN NATIONAL STANDARD IDENTIFICATION OF STATES.....FOR INFORM
DEPCC69                    (REF-MUJDC71)     =THE COPAK COMPRESSOR
WOWPS68                    (REF-PAMFS71)      (REF-SUSMS71) .=PROCEDURAL SEMANTICS FOR A QUESTION-ANSWERING MACHINE
SIRAE65    REF-MKRS71)     (REF-PAMFS71)      (REF-SUSMS71) .=ANSWERING ENGLISH QUESTIONS BY COMPUTER: A SURVEY (
WILPI53                    (REF-PAMFS71)     =PHILOSOPHICAL INVESTIGATIONS
WEUES66    Y               (REF-PAMFS71)     =EXPLORATIONS IN SEMANTIC THEOR
WOWSI67                    (REF-PAMFS71)     =SEMANTIC INTERPRETATION OF ENGLISH QUESTIONS ON A STRUCTURED DATA BASE
THJPS69                    (REF-PAMFS71)     =A PROGRAM FOR THE SYNTACTIC ANALYSIS OF ENGLISH SENTENCES
MOJNL69                    (REF-PAMFS71)     .=INFORMATION RETRIEVAL AND NATURAL LANGUAGE
PAMHC63                    (REF-PAMFS71)     =HOMOGRAPHS I: CLASSIFICATION AND IDENTIFICATION
STRIT68                    (REF-PAMFS71)     .=INTEGRATION OF TRANSFORMATIONAL THEORIES ON ENGLISH SYNTAX
SIRNL70    ERING SYSTEMS   (REF-PAMFS71)     =NATURAL LANGUAGE QUESTION ANSW
PAMCM67                    (REF-PAMFS71)     =COMPUTATIONAL MORPHOLOGY
SIRCM68                    (REF-PAMFS71)     =A COMPUTATIONAL MODEL OF VERBAL UNDERSTANDING
MOJPR69                    (REF-PAMFS71)     .=A PROGRESS REPORT ON THE USE OF ENGLISH IN INFORMATION RETRIEVAL
SAGAL68                    (REF-PAMFS71)     =AUTOMATED LANGUAGE PROCESSING
LASOS66     GRAMMAR        (REF-PAMFS71)     =AN OUTLINE OF STRATIFICATIONAL
SCRST67                    (REF-PAMFS71)     =STEPS TOWARD A MODEL OF LINGUISTIC PERFORMANCE
OTMIL69                    (REF-PAMFS71)     =INTERMEDIATE LANGUAGES FOR AUTOMATIC LANGUAGES PROCESSING
```

PRAIT69 (REF-PAMFS71) =IDENTIFICATION AND TRANSFORMATION OF TERMINAL MORPHEMES IN MEDICAL ENGLISH
SIRLA68 (REF-PAMFS71) =LINGUISTIC ANALYSIS OF CONSTRUCTED RESPONSES IN CAI
QURSM68 EF-KECCN71) (REF-PAMFS71) .=SEMANTIC MEMORY (R
OLYTD68 (REF-PAMFS71) =TOWARD THE DEVELOPMENT OF COMPUTATIONAL AIDS FOR OBTAINING A FORMAL DESCRIPTI
PRAAP69 (REF-PAMFS71) =AUTOMATIC PROCESSING OF MEDICAL ENGLISH
MOCLA69 (REF-PAMFS71) =LINGUISTICS AND AUTOMATED LANGUAGE PROCESSING
SCRTC69 (REF-PAMFS71) =TOWARDS A COMPUTATIONAL FORMALIZATION OF NATURAL LANGUAGE SEMANTICS
PAMHC67 (REF-PAMFS71) =HOMOGRAPHS II: CLASSIFICATION AND IDENTIFICATION
QURTL69 (REF-PAMFS71) (REF-SUSMS71) .=THE TEACHABLE LANGUAGE COMPREHENDER
MOCAL69 (REF-PAMFS71) =AUTOMATED LANGUAGE PROCESSING
KUSAS67 (REF-PAMFS71) =AUTOMATIC SYNTACTIC ANALYSIS
KAJSS65 EORY (REF-PAMFS71) =THE STRUCTURE OF A SEMANTIC TH
HAUDT64 (REF-PAMFS71) =DEPENDENCY THEORY: A FORMALIZATION AND SOME OBSERVATIONS
KAJRI67 RY (REF-PAMFS71) =RECENT ISSUES IN SEMANTIC THEO
HOCPU66 ANGUAGE (REF-PAMFS71) =THE PROBLEM OF UNIVERSALS IN L
HAZSL61 (REF-PAMFS71) =STRUCTURAL LINGUISTICS
KECNL68 (REF-PAMFS71) =A NATURAL LANGUAGE COMPILER FOR ON-LINE DATA MANAGEMENT
GAPLV63 (REF-PAMFS71) =A LINGUISTICS VIEW OF LANGUAGE DATA PROCESSING
COKES69 REF-SUSMS71) (REF-PAMFS71) .=EXPERIMENTS WITH A SEARCH ALGORITHM FOR THE DATA BASE OF A HUMAN BELIEF SYS
FIC CC68 (REF-PAMFS71) =THE CASE FOR CASE
BOUST68 (REF-PAMFS71) =SYNTACTIC THEORY IN COMPUTER IMPLEMENTATIONS
BOUAM65 (REF-PAMFS71) =THE ATOMIZATION OF MEANING
BAYOC SEMANTICS (REF-PAMFS71) =THE OUTLOOK FOR COMPUTATIONAL
BOUAL67 (REF-PAMFS71) =AUTOMATED LANGUAGE PROCESSING
BEAOP55 (REF-PAMFS71) =OPERATIONISM
CHPPM68 ANALYSIS (REF-PAMFS71) =A PROCEDURE FOR MORPHOLOGICAL
GAPPT67 (REF-PAMFS71) =PREDICATION TYPING: A PILOT STUDY IN SEMANTIC ANALYSIS;
FICTM66 (REF-PAMFS71) =TOWARD A MODERN THEORY OF CASE
COLSN65 ATHOLOGY (REF-PAMFS71) =SYSTEMATIZED NOMENCLATURE OF P
ALWPL64 (REF-PAMFS71) =PHILOSOPHY OF LANGUAGE
ABRIM69 (REF-PAMFS71) =IMPLICATIONAL MOLECULES: A METHOD FOR EXTRACTING MEANING FROM INPUT SENTENCES
CESMT (REF-PAMFS71) =MECHANICAL TRANSLATION: THE CORRELATIONAL SOLUTION
CHNRN (REF-PAMFS71) .=REMARKS ON NORMALIZATION
EALAU67 (REF-PAMFS71) =AUTOMATIC DETERMINATION OF PARTS OF SPEED OF ENGLISH WORDS
GLEMP70 (REF-PAMFS71) =THE MULTISTORE PARSER FOR HIERARCHICAL SYNTACTIC STRUCTURES
CHNAT65 (REF-LOBHF71) (REF-PAMFS71) .=ASPECTS OF THE THEORY OF SYNTAX (REF-LOUCC71)
SUSST69 (REF-SUSMS71) =A SEMANTIC THEORY BASED UPON INTERACTIVE MEANING
SHSNS69 (REF-SUSMS71) =A NET STRUCTURE BASED RELATIONAL QUESTION ANSWERER: DESCRIPTION AND EXAMPLES
SUSDR69 NERATOR (REF-SUSMS71) =A DIRECTED RANDOM PARAGRAPH GE
SIRCM68 (REF-SUSMS71) =A COMPUTATIONAL MODEL OF VERBAL UNDERSTANDING
SIRAE65 REF-PAMFS71) (REF-SUSMS71) .=ANSWERING ENGLISH QUESTIONS BY COMPUTER: A SURVEY (REF-MKRS71) (
WOWPS68 REF-PAMFS71) (REF-SUSMS71) .=PROCEDURAL SEMANTICS FOR A QUESTION-ANSWERING MACHINE (
YOMIS69 (REF-SUSMS71) =INFORMATION SCIENCE: TOWARD THE DEVELOPMENT OF A TRUE SCIENTIFIC DISCIPLINE
WAUCL69 (REF-SUSMS71) =COMPUTATIONAL LINGUISTIC TECHNIQUES IN AN ON-LINE SYSTEM FOR TEXTUAL ANALYSIS
SAGEP70 (REF-SUSMS71) =EVALUATION PROBLEMS IN INTERACTIVE INFORMATION RETRIEVAL
WEPCR71 TRUCTURE (REF-SUSMS71) =CYLINDERS: A RELATIONAL DATA S
MIMSI69 G (REF-SUSMS71) =SEMANTIC INFORMATION PROCESSIN
QURTL69 (REF-PAMFS71) (REF-SUSMS71) .=THE TEACHABLE LANGUAGE COMPREHENDER
SIMNL70 (REF-SUSMS71) (REF-MKRS71) .=NATURAL LANGUAGE QUESTION-ANSWERING SYSTEMS.
LOCMS66 (REF-SUSMS71) =MEMORY STRUCTURES IN DEACON NATURAL LANGUAGE QUESTION-ANSWERING SYSTEM
LOPRE69 SYSTEM (REF-SUSMS71) =A RAPIDLY EXTENSIBLE LANGUAGE
OVLLA69 REVIEW (REF-SUSMS71) =LIBRARY AUTOMATION:A CRITICAL
KAMMS70 (REF-SUSMS71) =THE MIND SYSTEM: THE STRUCTURE OF THE SEMANTIC FILE
KOMAQ69 (REF-SUSMS71) =AUTOMATIC QUESTION-ANSWERING OF ENGLISH-LIKE QUESTIONS ABOUT SIMPLE DIAGRAMS
HIDLR69 (REF-SUSMS71) .=THE LEADER RETRIEVAL SYSTEM
KAADR67 (REF-SUSMS71) (REF-KUJQ071).=DATA-RETRIEVAL BY COMPUTER: A CRITICAL SURVEY
KEGNL68 (REF-SUSMS71) .=A NATURAL LANGUAGE COMPILER FOR ON-LINE DATA MANAGEMENT

```
GRCAT69                          (REF-SUSMS71)     =APPLICATION OF THEOREM PROVING TO PROBLEM SOLVING
CHECS69                          (REF-SUSMS71)     =COMPUTER SOLUTION OF CALCULUS WORD PROBLEMS
AUCME68                          (REF-SUSMS71)     =MEDLARS 1963-1967
CORIS69        VAL               (REF-SUSMS71)     =INFORMATION STORAGE AND RETRIE
ABRIM69                          (REF-SUSMS71)     =IMPLICATIONAL MOLECULES: A METHOD FOR EXTRACTING MEANING FROM INPUT SENTENCES
BOUQA64         (REF-KUJQQ71)    (REF-SUSMS71)     (REF-LODCC71).=A QUESTION-ANSWERING SYSTEM FOR HIGH SCHOOL ALGEBRA WORK PROBLE
COLOL68                          (REF-SUSMS71)     =AN ON-LINE QUESTION-ANSWERING SYSTEM WITH NATURAL LANGUAGE AND PICTORIAL INPU
COKES69        IEF SYSTEM        (REF-SUSMS71)      (REF-PAMFS71) .=EXPERIMENTS WITH A SEARCH ALGORITHM FOR THE DATA BASE OF A HU
BRWGP66                          (REF-SUSMS71)     =GUIDE TO THE PROCESSING, STORAGE AND RETRIEVAL OF BIBLIOGRAPHIC INFORMATION A
VAPCC70                          (REF-TEDAR71)     =CAN COBOL COPE
TUSD63         IGN               (REF-TEDAR71)     =SEQUENTIAL DATA PROCESSING DES
YOJAF59                          (REF-TEDAR71)     =ABSTRACT FORMULATION OF DATA PROCESSING PROBLEMS
TEDMD69                          (REF-TEDAR71)     =METHODOLOGY FOR THE DESIGN OF INFORMATION PROCESSING SYSTEMS
MAJDR67        YSTEMS            (REF-TEDAR71)     =DESIGN OF REAL-TIME COMPUTER S
NACAD67                          (REF-TEDAR71)     =ACCURATELY DEFINED SYSTEMS
LABTA66                          (REF-TEDAR71)     =THEORETICAL ANALYSIS OF INFORMATION SYSTEMS
GRMSG63                          (REF-TEDAR71)     =SYSTEMS GENERATION OUTPUT DECOMPOSITION METHOD
IBMTA68                          (REF-TEDAR71)     =THE TIME AUTOMATED GRID SYSTEM (TAG): SALES AND SYSTEMS GUIDE
KIPSA69                          (REF-TEDAR71)     =SYSTEMS ANALYSIS DOCUMENTATION: COMPUTER AIDED DATA DIRECTORY DEFINITION
GRCPI69                          (REF-TEDAR71)     =PRINCIPLES OF THE IDENTIFICATION OF INFORMATION
GRCSI70                          (REF-TEDAR71)     =SPECIFICATION AND INTERROGATION OF FORMAL CONTROL SYSTEMS
CODIA62         REPORT           (REF-TEDAR71)     =AN INFORMATION ALGEBRA PHASE I
SEMFD69                          (REF-WACQE71)     =FILE DESIGN HANDBOOK
TEURS69        NIQUES            (REF-WACQE71)     =A REVIEW OF SYSTEM DESIGN TECH
SEMFO68                          (REF-WACQE71)     =A FILE ORGANIZATION EVALUATION MODEL-FOREM
LASA63                           (REF-WACQE71)     =SOME APPROACHES TO THE THEORY OF INFORMATION SYSTEMS
NOJDO61                          (REF-WACQE71)     =ON THE DESIGN AND OPTIMIZATION OF INFORMATION PROCESSING SYSTEMS
MCWTA69        YSTEMS            (REF-WACQE71)     =TOWARD THE AUTOMATIC DESIGN OF DATA ORGANIZATION FOR LARGE SCALE INFORMATION
AFCFF67        CHNIQUES          (REF-WACQE71)     =FORMATTED FILE ORGANIZATION TE
UAOE65                           (REF-WACQE71)     =ON OPTIMAL EXTRACING FROM A MULTIPLE FILE DATA STORAGE SYSTEM : AN APPLICATIO
BUWFO63        NG                (REF-WACQE71)     =FILE ORGANIZATION AND ADDRESSI
GHSFO69        ENTIAL FILES      (REF-WACQE71)     (REF-KNGHT71).=FILE ORGANIZATION: ON THE SELECTION OF RANDOM ACCESS INDEX POIN
WIAMC70                          (REF-WIAFS71)     =A METHODOLOGY FOR COMPARISON OF FILE ORGANIZATION AND PROCESSING PROCEDURES F
MALMF69                          (REF-WIAFS71)     =A MODEL FOR FILE STRUCTURE DETERMINATION FOR LARGE ON-LINE DATA FILES
LEUFS69        STEMS             (REF-WIAFS71)     =FILE STRUCTURES FOR ON-LINE SY
CHNCF69                          (REF-WIAFS71)     =A COMPARISON OF FILE ORGANIZATION TECHNIQUES
EISP68                           (REF-WIAFS71)     =SYSTEMS PROGRAMMING DOCUMENTATION, TABLES, REMOTE FILE MANAGEMENT SYSTEM, PHA
EVUDS71                          (REF-WIAFS71)     =DATA STRUCTURES AND ALGORITHMIC DETERMINACY
FLIAI61        SORTING           (REF-WIAFS71)     =ANALYSIS OF INTERNAL COMPUTER
UAGI68                           (REF-WIAFS71)     =GENERAL INFORMATION, REMOTE FILE MANAGEMENT SYSTEM, PHASE I
CHYFT71        EW LEGAL TEXTS    (REPORT           ON THE FIRST PHASE OF THE RESPONSA RETRIEVAL PROJECT).=FULL TEXT DOCUMENT RETR
WHAPM50        IA MATHEMATICA    (SECOND           EDITION) (REF-KUJQQ71).=PRINCIP
IBMTA68        ED GRID SYSTEM    (TAG)              SALES AND SYSTEMS GUIDE (REF-TEDAR71).=THE TIME AUTOMAT
AUCME68              MEDLARS     1963-1967          (REF-SUSMS71).=
AFICP69                AFIPS     1969              SPRING JOINT COMPUTER CONFERENCE (REF-MKRS71).=
MKRS71         CTIVES FOR THE    1971              ACM INFORMATION STORAGE AND RETRIEVAL SYMPOSIUM .=INTRODUCTION AND PERSPE
CHWTR62        S COUCHED IN A    'SEMIFORMAL'      ENGLISH-LIKE LANGUAGE. (REF-MKRS71).=TRANSLATION OF RETRIEVAL REQUEST
```

ABRIM69 ABELSON,R. CARROLL,C.
IMPLICATIONAL MOLECULES: A METHOD FOR EXTRACTING MEANING FROM INPUT SENTENCES (REF-PAMFS71).=
PROC. 1969 INT. CONF. ON ART. INTELL.

ABRIM69 ABELSON,R. REICH,C.
IMPLICATIONAL MOLECULES: A METHOD FOR EXTRACTING MEANING FROM INPUT SENTENCES (REF-SUSMS71).=
PROC. INT. JT. CONF. ART. INTEL., 1969, PP.641-647

ACWSC54 ACKERMANN,W.
SOLVABLE CASES OF THE DECISION PROBLEM (REF-DIRRD71).=
STUDIES IN LOGIC AND THE FOUN. OF MATH. 1954

ADGAO ADEL'SON-VEL'SKIJ,G. LANDIS,Y.
AN ALGORITHM FOR THE ORGANIZATION OF INFORMATION (REF-KNGBT71).=
DAKLADY AKADAMII NAUK USSR, VOL.16, #2, PP.163-166

AFCFF67 AF CONTRACT 30602-4088
FORMATTED FILE ORGANIZATION TECHNIQUES (REF-WACQE71).=
FINAL REPORT, CONTRACT AF 30602-4088, MAY 1967

AFICP69 AFIPS
AFIPS 1969 SPRING JOINT COMPUTER CONFERENCE (REF-MKRS71).=
AFIPS 1969 SJCC , P. 457-490

ALWPL64 ALSTON,W.
PHILOSOPHY OF LANGUAGE (REF-PAMFS71).=
PRENTICE-HALL, INC., 1964

AQLNA65 AQVIST,L.
A NEW APPROACH TO THE LOGICAL THEORY OF INTERROGATIVES.
PART I, ANALYSIS (REF-KUJQQ71).=
UNIVERSITY OF UPPSALA, UPPSALA, SWEDEN, 1965

ARJIS69 ARON,J.
INFORMATION SYSTEMS IN PERSPECTIVE (REF-GURER71).=
COMPUTING SURVEYS 1, DEC, 1969, PP. 213-236

ASRIT65 ASH,R.
INFORMATION THEORY (REF-KAAEU71).=
PP. 27-43, INTERSCIENCE PUBLISHERS, NEW YORK, 1965

ASWIR68 ASH,W SIBLEY,E.
TRAMP: AN INTERPRETIVE ASSOCIATIVE PROCESSOR WITH DEDUCTIVE CAPABILITIES (REF-KECCN71).=
PROC. 23RD NAT. ACM CONF. , PP.143-156 1968

AUCME68 AUSTIN,C.
MEDLARS 1963-1967 (REF-SUSMS71).=
NATIONAL LIBRARY OF MEDICINE, BETHESDA, MD., 1968

AUSIM64 AUSTIN, C.J.

THE MEDLARS SYSTEM. (REF-MKRS71).=
DATAMATION, 10:12 (DEC. 1964) P. 28-31.

BAYOC BAR-HILLEL,Y.
THE OUTLOOK FOR COMPUTATIONAL SEMANTICS (REF-PAMFS71).=
PROC. OF THE CONF. ON COMP. RELATED SEMANTIC ANAL.

BEAOP55 BENJAMIN,A.
OPERATIONISM (REF-PAMFS71).=
CHARLES C. THOMAS, SPRINGFIELD, INC., 1955

BEJIS63 BECKER,J. HAYES,R.
INFORMATION STORAGE AND RETRIEVAL: TOOLS, ELEMENTS, THEORIES
(REF-GURER71).=
WILEY, 1963

BEJOA69 BENNETT,J.
ON-LINE ACCESS TO INFORMATION: NSF AS AN AID TO THE INDEXER/
CATALOGER (REF-CHYFT71).=
AMER. DOCUMENTATION 20, PP.213-220, 1969

BENAQ63 BELNAP,N.,JR.
AN ANALYSIS OF QUESTIONS: PRELIMINARY REPORT
(REF-KUJQQ71).=
SYSTEM DEVELOPMENT CORPORATION, TM-1287/000/00 JUNE 3, 1963

BOAEN60 BOOTH,A COLIN,H.
ON THE EFFICIENCY OF A NEW METHOD OF DICTIONARY CONSTRUCTION
(REF-KNGBT71).=
INFORMATION AND CONTROL, VOL 3, PP. 334-341, DEC. 1960

BOCMI63 BOURNE,C.
METHODS OF INFORMATION HANDLING (REF-GURER71).=
WILEY, 1963

BODAL67 BOBROW,D. FRASIER,J. QUILLIAN,M.
AUTOMATED LANGUAGE PROCESSING (REF-PAMFS71).=
REVIEW OF INFOR. SCI. AND TECH., VOL. 2, 1967

BODAM65 BOLLINGER,D.
THE ATOMIZATION OF MEANING (REF-PAMFS71).=
LANGUAGE, VOL.JU, NO. 4, 1965

BODQA64 BOBROW,D.
A QUESTION-ANSWERING SYSTEM FOR HIGH SCHOOL ALGEBRA WORK
PROBLEMS (REF-KUJQQ71) (REF-SUSMS71) (REF-LODCC71).=
AFIPS CONFERENCE PROCEEDINGS VOL. 26 (1964 FJCC) P. 591-614

BODST68 BOBROW,D.
SYNTACTIC THEORY IN COMPUTER IMPLEMENTATIONS
(REF-PAMFS71).=
AUTO. LANG. PROCESSING, 1968

BRWGP66 BRANDHORST,W. ECKERT,P
GUIDE TO THE PROCESSING, STORAGE AND RETRIEVAL OF
BIBLIOGRAPHIC INFORMATION AT THE NASA SCIENTIFIC AND
TECHNICAL INFORMATION FACILITY (REF-SUSMS71).=

DOCUMENTATION INC., NASA CR-62033, 1966

BUVAW45 BUSH, VANNEVAR
AS WE MAY THINK (REF-MKRS71).=
THE ATLANTIC MONTHLY 176,(JULY 1945) P.101-108.

BUWF063 BUCHOLZ,W.
FILE ORGANIZATION AND ADDRESSING (REF-WACQE71).=
IBM SYSTEMS JOUR. 2, PP. 86-111, JUNE 1963

CAJUP65 CANTER,J. DONAGHEY,C.
UPLIFTS-UNIVERSITY OF PITTSBURGH LINEAR FILE TANDEM SYSTEM
(REF-GURER71).=
COMM. ACM 8, SEPT. 1965, PP. 579-581

CARIS58 CARNAP,R.
INTRODUCTION TO SYMBOLIC LOGIC AND ITS APPLICATIONS
(REF-KUJQQ71).=
DOVER PUBLICATIONS, INC., NEW YORK, 1958

CARLS37 CARNAP,R.
THE LOGICAL SYNTAX OF LANGUAGE,
(REF-KUJQQ71).=
ROUTLEDGE AND KEGAN PAUL, LTD., LONDON, 1937, (ENGLISH ED.)

CARMN56 CARNAP,R.
MEANING AND NECESSITY (REF-KUJQQ71) (REF-KECCN71) .=
2ND EDITION UNIVERSITY OF CHICAGO PRESS, CHICAGO, 1956

CESMT CECCATO,S. ET AL
MECHANICAL TRANSLATION: THE CORRELATIONAL SOLUTION
(REF-PAMFS71).=
AD-409,07

CGMIE58 CARLSEN, R. D. GERNER, N.H. MARSHALL, H.S.
INFORMATION CONTROL (REF-MKRS71).=
NORTH AMERICAN AVIATION, CANOGA PARK, CALIFORNIA. 1958.

CHBCT65 CHEYDLEUR,B. (ED)
COLLOQUIUM ON TECHNICAL PRECONDITIONS FOR RETRIEVAL CENTER
OPERATIONS (REF-GURER71).=
SPARTAN, 1965

CHBSH63 CHEYDLEUR,B.
SHIEF: A REALIZABLE FORM OF ASSOCIATIVE MEMORY
(REF-GURER71).=
AM. DOCUMENTATION, JAN. 1963, PP. 56-57

CHCIT61 CHERRY,C.
INFORMATION THEORY (REF-GURER71).=
BUTTERWORTH, 1961

CHECS69 CHARNIAK,E.
COMPUTER SOLUTION OF CALCULUS WORD PROBLEMS
(REF-SUSMS71).=
PROC. INT. JT. CONF. ART. INTEL., 1969, PP.303-316

CHNAT65 CHOMSKY,N.
ASPECTS OF THE THEORY OF SYNTAX
(REF-LOUCC71) (REF-DOBHF71) (REF-PAMFS71) .=
MIT PRESS, 1965

CHNCF69 CHAPIN,N.
A COMPARISON OF FILE ORGANIZATION TECHNIQUES
(REF-WIAFS71).=
PROCEEDINGS OF THE 24TH NATIONAL CONFERENCE ACM PP.273-283

CHNRN CHOMSKY,N
REMARKS ON NORMALIZATION (REF-PAMFS71) .=
READING IN ENGLISH TRANSFORMATIONAL GRAMMAR

CHPPM68 CHAPIN,P.
A PROCEDURE FOR MORPHOLOGICAL ANALYSIS (REF-PAMFS71).=
INFOR. SYS. LANG. STUDIES, #18, THE MITRE CROP., 1968

CHTGI68 CHEATHAM,T.E.
ON THE GENERATION AND IMPLEMENTATION OF REDUCTIONS ANALYSIS
PROGRAMS FOR CONTEXT FREE GRAMMAR (REF-LOUCC71).=
MASS. COMP. ASSO., CA-6902-2911, FEB. 1968

CHWIR62 CHEATHAM, T. WARSHALL, J.
TRANSLATION OF RETRIEVAL REQUESTS COUCHED IN A 'SEMIFORMAL'
ENGLISH-LIKE LANGUAGE. (REF-MKRS71).=
COMM. ACM 5:1 (JAN. 1962) P. 36-38.

CHYAG66 CHOUEKA,Y.
AUTOMATIC GRAMMATICAL ANALYSIS OF THE HEBREW VERB (IN
HEBREW) (REF-CHYFT71).=
PROC. 2ND NAT. CONF. ,INFOR. PROC. ASSO.,ISRAEL, P.49-67, 19

CHYAG67 CHOUEKA,Y.
AUTOMATIC GRAMMATICAL ANALYSIS OF THE HEBREW VERB (IN
HEBREW) ENGLISH ABSTRACT (REF-CHYFT71).=
THE FINITE STRING 4,1967

CHYFT71 CHOUEKA,Y. COHEN,M. DUECK,J. FRAENKEL,A. SLAE,M.
FULL TEXT DOCUMENT RETRIEVAL: HEBREW LEGAL TEXTS (REPORT ON
THE FIRST PHASE OF THE RESPONSA RETRIEVAL PROJECT).=
ACM SYMP. ON IS&R, APRIL 1971

CLCFD66 CLEVERDON,C. KEEN,M.
FACTORS DETERMINING THE PERFORMANCE OF INDEXING SYSTEMS
(REF-JADOC71).=
ASLIB CRANFIELD RESEARCH PROJECT, 1966, VOLS. 1 AND 2

CODIA62 CODASYL DEVELOPMENT COMMITTEE
AN INFORMATION ALGEBRA PHASE I REPORT (REF-TEDAR71).=
COMMUNICATIONS ACM, 5,4, APRIL 1962, P. 190-204

CODSG69 CODASYL
A SURVEY OF GENERALIZED DATA BASE MANAGEMENT SYSTEMS
(REF-MKRS71) .=
CODASYL SYS, COMM. TECH. REP., ACM, 1969

COJEM69 CONSIDINE,J. WEIS,A.
ESTABLISHMENT AND MAINTENANCE OF A STORAGE HIERARCHY FOR AN ON-LINE DATA BASE UNDER TSS/360 (REF-KAAEU71).=
PROC. 1969 AFIPS FJCC, VOL.35, PP.433-440

COKES69 COLBY,K. TESLER,L. ENEA,H.
EXPERIMENTS WITH A SEARCH ALGORITHM FOR THE DATA BASE OF A HUMAN BELIEF SYSTEM (REF-SUSMS71) (REF-PAMFS71) .=
PROC. 1969 INT. CONF. ON ART. INTELL.

COLOL68 COLES,L.
AN ON-LINE QUESTION-ANSWERING SYSTEM WITH NATURAL LANGUAGE AND PICTORIAL INPUT (REF-SUSMS71).=
PROC. ACM 23RD NAT. CONF.,1968, PP.157-167

COLSN65 COLLEGE OF AMERICAN PATHOLOGISTS
SYSTEMATIZED NOMENCLATURE OF PATHOLOGY (REF-PAMFS71).=
COLLEGE OF AMERICAN PATHOLOGISTS, 1965

COMMS70 COHEN,M.
FULL TEXT INFORMATION RETRIEVAL (IN HEBREW) (REF-CHYFT71).=
M.SC. THESIS SUBMITTED TO TEL-AVIV UNIVERSITY, 1970

CORIS69 CORNELL UNIVERSITY
INFORMATION STORAGE AND RETRIEVAL (REF-SUSMS71).=
SCIENTIFIC REP. #ISR-16, CORNELL UNIV., 1969

CUPAR69 CULICOVER,P. KIMBALL,J. LEWIS,C. LOVEMAN,D.
AN AUTOMATED RECOGNTION GRAMMAR FOR ENGLISH (REF-LODCC71).=
IBM, FSC-69-5007, JULY 1969, 103P.

DAG168 DALE, A.G. MIRIAM EICHELKRAUT
GENERAL INFORMATION, REMOTE FILE MANAGEMENT SYSTEM, PHASE I (REF-WIAFS71).=
TSD-0, UNIV. OF TEXAS AT AUSTIN COMPUTER CERTER, 1968

DAOE65 DAY,R.
ON OPTIMAL EXTRACING FROM A MULTIPLE FILE DATA STORAGE SYSTEM : AN APPLICATION OF INTEGER PROGRAMMING (REF-WACQE71).=
JOUR. OP. RES. 13, 3, PP. 482-94, MAY-JUNE, 1965

DEAPS65 DEACON PROJECT
PHRASE-STRUCTURE ORIENTED TARGETING QUERY LANGUAGE (REF-LODCC71).=
RM65TMP-64, TEMPO, GENERAL CO., 1965

DEPCC69 DEMAINE,P. SPRINGER,G.
THE COPAK COMPRESSOR (REF-MUJDC71).=
SELETED PAPERS FROM IAG CONF. 1969, PP.149-158

DIRRD71 DIPAOLA,R.
THE RELATIONAL DATA FILE AND THE DECISION PROBLEM FOR CLASSES OF PROPER FORMULAS.=
ACM SYMP. ON IS&R, APRIL 1971

DIRRU69 DIPAOLA,R.
THE RECURSIVE UNSOLVABILITY OF THE DECISION PROBLEM FOR THE CLASS OF DEFINITE FORMULAS (REF-DIRRU71).=
JACM, VOL. 16, NO. 2, APRIL 1969, PP. 324-327

DOBHF71 DOSTERT,B. THOMPSON,F.
HOW FEATURES RESOLVE SYNTACTIC AMBIGUITY.=
ACM SYMP. ON IS&R, APRIL 1971

DOGED69 DODD,G.
ELEMENTS OF DATA MANAGEMENT SYSTEMS (REF-GURER71).=
COMPUTING SURVEYS 1, JUNE, 1969, 117-133

DOGRE69 DOSERT,G. THOMPSON,F.
A RAPIDLY EXTENSIBLE LANGUAGE SYSTEM: REL ENGLISH (REF-DOBHF71).=
1969 INT. CONF. ON COMP. LING. NO.35 37P.

DOJ DONNE,J.
DEVOTIONS (REF-DOBHF71).=
UNKNOWN

DOLIA62 DOYLE,L.
INDEXING AND ABSTRACTING BY ASSOCIATION (REF-JADOC71).=
AMERICAN DOCUMENTATION, 13, 1962, PP. 378-390

DOLSR DOYLE,L.
SEMANTIC ROAD MAPS FOR LITERATURE SEARCHERS (REF-JADOC71).=
J.ACM., 8, 1961, PP. 553-578

DZBOS65 DZUBAK,B. WARBURTON,B.
THE ORGANIZATION OF STRUCTURED FILES (REF-GURER71).=
COMM. ACM 8, JULY, 1965, 446-452

EAJGR65 EARLY,J.
GENERATING A RECOGNIZER FOR A BNF GRAMMAR (REF-DOBHF71).=
TECH. REP., CARNEGIE INST. OF TECH. 1965

EALAD67 EARL,L.L.
AUTOMATIC DETERMINATION OF PARTS OF SPEED OF ENGLISH WORDS (REF-PAMFS71).=
MECH. TRANS. AND COMP. LING., VOL. 1, #3,4, U. OF CHICAGO

EISP68 EICHELKRAUT, MIRIAM
SYSTEMS PROGRAMMING DOCUMENTATION, TABLES, REMOTE FILE MANAGEMENT SYSTEM, PHASE 1 (REF-WIAFS71).=
TSD-1, UNIV. OF TEXAS AT AUSTIN COMPUTER CENTER, 1968

ELW CT63 ELDRIDGE,W. DENNIS, S.
THE COMPUTER AS A TOOL FOR LEGAL RESEARCH (REF-CHYFT71).=
LAW AND CONTEMPORARY PROBLEMS 28, PP. 78-99, 1963

EVDDS71 EVERTT,D.G.
DATA STRUCTURES AND ALGORITHMIC DETERMINACY

(REF-WIAFS71).=
PH.D. DISSERTATION, UNIV. OF TEXAS AT AUSTIN, 1971

EVTHP64 EVANS,T.
A HEURISTIC PROGRAM TO SOLVE GEOMETRIC-ANALOGY PROBLEMS
(REF-JADOC71).=
PROC. 1964 AFIPS SJCC PP.327-338

FART161 FANO,R.
TRANSMISSION OF INFORMATION (REF-KAAEU71).=
MIT PRESS, CAMBRIDGE, MASS., 1961

FEAF158 FEINSTEIN,A.
FOUNDATIONS OF INFORMATION THEORY (REF-KAAEU71).=
MCGRAW-HILL BOOK CO., NEW YOUR, 1958

FEDBT68 FERGUSON,D.
BALANCED TREE SEARCHING (REF-KNGBT71).=
INTERNAL DOCUMENT, PROGRAMMATICS, INC., JAN. 1968

FEJAB69 FELDMAN,J. ROVNER,P.
AN ALGOL-BASED ASSOCIATE PROCESSOR (REF-KECCN71).=
ACM COMM. VOL 12/8 AUG 1969, PP. 439-449

FERDS69 FEYS,R. FITCH,F. (ED)
DICTIONARY OF SYMBOLS OF MATHEMATICAL LOGIC
(REF-KUJQQ71).=
NORTH-HOLLAND PUBLISHING CO., AMSTERDAM, 1969

FEW1P68 FELLER,W.
AN INTRODUCTION TO PROBABILITY THEORY AND ITS APPLICATIONS
(REF-GUREK71).=
WILEY, 1968

FIC CC68 FILLMORE,C.
THE CASE FOR CASE (REF-PAMFS71).=
UNIVERSALS IN LINGUISTICS THEORY, 1968

FICIM66 FILLMORE,C.
TOWARD A MODERN THEORY OF CASE (REF-PAMFS71).=
REP. #13, THE OHIO STATE U., ,1966

FIJGF70 FINKE,J.
A GENERALIZED FILE MANAGEMENT SYSTEM (REF-HUEHM71).=
U. W. COMP. SCI. M.S. THESIS, 1970A

FIJUG70 FINKE,J.
A USERS GUIDE TO THE OPERATION OF AN INFORMATION STORAGE AND
RETRIEVAL SYSTEM ON THE B5500 COMPUTER (REF-HUEHM71).=
U. W. COMP.SCI. TECH. REPORT 70-1-3, 1970B

FLIAI61 FLORES,I.
ANALYSIS OF INTERNAL COMPUTER SORTING (REF-WIAFS71).=
JACM 8, 1961, PP.41-80

FOCIS65 FOSTER,C.
INFORMATION STORAGE AND RETRIEVAL USING AVL TREES

(REF-KNGBT71).=
PROC. 20TH NAT. ACM CONF. PP.192-205 1965

FRAC169 FRAENKEL,A.
THE COMPUTER AND INFORMATION RETRIEVAL AS AUXILIARY TOOLS IN HEBREW LEGAL RESEARCH (IN HEBREW) (REF-CHYFT71).=
TECH. REP., WEIZMANN INST. OF SCI. ,BAR-ILAN U. 1969

FRALI68 FRAENKEL,A.
LEGAL INFORMATION RETRIEVAL (REF-CHYFT71).=
ADVANCES IN COMPUTERS 9, PP. 113-178, 1968

FRAMR68 FRAENKEL,A.
MODERN RETRIEVAL METHODS APPLIED TO JEWISH CASE LAW
(REF-CHYFT71).=
PROC. 1ST INT. SYMP. SOC. TECH. WRITERS & PUBL. 1968, ISREAL

FRCSE52 FRIES,C.
THE STRUCTURE OF ENGLISH (REF-DOHHF71).=
HARCOURT, BRACE AND WORLD, 1952, 304 P.

GAHCR56 GAINES,H.
CRYPTANALYSIS (REF-MUJDC71).=
DOVER, NEW YORK, 1956

GAPLV63 GARVIN,P.
A LINGUISTICS VIEW OF LANGUAGE DATA PROCESSING
(REF-PAMFS71).=
NATURAL LANGUAGE AND THE COMPUTER, 1963

GAPNL63 GARVIN,P. (ED.)
NATURAL LANGUAGE AND THE COMPUTER (REF-GURER71).=
MCGRAW-HILL, 1963

GAPPT67 GARVIN,P.
PREDICATION TYPING: A PILOT STUDY IN SEMANTIC ANALYSIS;
(REF-PAMFS71).=
LANGUAGE, SUPPLEMENT 43 (2), PT. II, 1967

GHSAF68 GHOSH,S. ABRAHAM,C.
APPLICATION OF FINITE GEOMETRY IN FILE ORGANIZATION FOR RECORDS WITH MULTIPLE-VALUED ATTRIBUTES (REF-GURER71).=
IBM J. RES. DEVELOP. 12 (MARCH, 1968), 180-187

GHSFO69 GHOSH,S. SENKO,M.
FILE ORGANIZATION: ON THE SELECTION OF RANDOM ACCESS INDEX POINTS FOR SEQUENTIAL FILES (REF-WACQE71) (REF-KNGBT71).=
JACM, VOL. 16, NO. 4, PP 569-579, OCT 1969

GLEMP70 GLASERFELT, E. PISONI,P.
THE MULTISTORE PARSER FOR HIERARCHICAL SYNTACTIC STRUCTURES
(REF-PAMFS71).=
COMMUNOCATIONS OF THE ACM, VIL. 13, NO. 2, FEB. 1970

GOSBP65 GOLOMB,S BAUMERT,L.
BACKTRACK PROGRAMMING (REF-GURER71).=
J. ACM 12. (OCT. 1965), 516-524

GRCAT69 GREEN,C.
APPLICATION OF THEOREM PROVING TO PROBLEM SOLVING
(REF-SUSMS71).=
PROC. INT. JT. CONF. ART. INTEL., 1969, PP.219-239

GRCPI69 GRINDLEY,C. STEVENS,W.
PRINCIPLES OF THE IDENTIFICATION OF INFORMATION
(REF-TEDAR71).=
FILE ORGANIZ.,IAG PUBL., NO. 3, PP60-68, 1969

GRCSI70 GRINDLEY,C.
SPECIFICATION AND INTERROGATION OF FORMAL CONTROL SYSTEMS
(REF-TEDAR71).=
TIMES XVII CONF. JULY 1970

GRMSG63 GROSZ,M.
SYSTEMS GENERATION OUTPUT DECOMPOSITION METHOD
(REF-TEDAR71).=
STANDARD OIL COMPANY OF NEW JERSEY, JULY 1963

GSRRC69 GUSTAFSON,R.
A RANDOMIZED COMBINATORIAL FILE STRUCTURE FOR STORAGE AND
RETRIEVAL SYSTEMS (REF-GURER71).=
PH.D. THESIS, U. OF SOUTH CAROLINA, 1969

GULEI63 GUTENMAKHER,L.
ELECTRONIC INFORMATION-LOGIC MACHINES (REF-GUREP71).=
INTERSCIENCE, 1963

GURER71 GUSTAFSON,R.
ELEMENTS OF THE RANDOMIZED COMBINATORIAL FILE STRUCTURE.=
ACM SYMP. ON IS&R, APRIL 1971

GWCLB61 GREEN, B. F. WOLF,A.K. CHOMSKY,C. LAUGHERY,K.
BASEBALL: AN AUTOMATIC QUESTION-ANSWERER.
(REF-MKRS71).=
PROCEEDINGS OF THE WESTERN JOINT COMPUTER CONFERENCE, 1961

HADCL63 HARRAH,D.
COMMUNICATION: A LOGICAL MODEL (REF-KUJQQ71).=
MIT PRESS, CAMBRIDGE, MASS., 1963

HADUT64 HAYS,D.
DEPENDENCY THEORY: A FORMALIZATION AND SOME OBSERVATIONS
(REF-PAMFS71).=
LANGUAGE, VOL. 40, NO. 9, 1964

HADIC67 HAYS,D.
INTRODUCTION TO COMPUTATIONAL LINGUISTICS
(REF-GURER71).=
ELSEVIER, 1967

HAGGG HARMON,G.
GENERATIVE GRAMMARS WITHOUT TRANSFORMATIONAL RULES: A
DEFENSE OF PHRASE STRUCTURE (REF-UOBHF71).=
LANGUAGE, VOL. 39, PP 597-616

HAJMT67 HARTIGAN,J.A.
REPRESENTATION OF SIMILARITY MATRICES BY TREES
(REF-JADOC71).=
JACM 62, 1967, PP.1140-1158

HAMAC63 HANAN,M. DALERMO,F.
AN APPLICATION OF CODING THEORY TO A FILE ADDRESS PROBLEM
(REF-GURER71).=
IBM J. RES. DEVELOP., APRIL, 1963, 127-129

HAZSL61 HARRIS,Z.
STRUCTURAL LINGUISTICS (REF-PAMFS71).=
THE UNIVERSITY OF CHICAGO PRESS, 1961

HIDLR69 HILLMAN,D. KASARDA,A.
THE LEADER RETRIEVAL SYSTEM (REF-SUSMS71) .=
PROC. AFIPS 1969 SJCC, PP. 447-455

HITSC62 HIBBARD,T.
SOME COMBINATORIAL PROPERTIES OF CERTAIN TREES
(REF-KNGBT71).=
JACM, VOL 9, NO.1, PP. 13-28, JAN, 1962

HOCPU66 HOCKETT,C.
THE PROBLEM OF UNIVERSALS IN LANGUAGE (REF-PAMFS71).=
UNIVERSALS OF LANGUAGE, 1966

HOJSS62 HORTY,J.
SEARCHING STATUTORY LAW BY COMPUTER (REF-CHYFT71).=
REP. #2, HEALTH LAW CENTER, UNIV. OF PITTSBURGH 1962

HSDFS70 HSIAO,D. HAVARY,F.
A FORMAL SYSTEM FOR INFORMATION RETRIEVAL FROM FILES
(REF-GURER71).=
COMM. ACM 13, FEB, 1970, 67-73

HUDMC HUFFMAN,D.
A METHOD FOR THE CONSTRUCTION OF MINIMUM REDUNDANCY CODES
(REF-KAAEU71).=
PROC. IRE, VOL.40, NO.10 , 1952, PP.1098-1101

HUECS70 HUNT,E.
THE COMPUTER SCIENCE TEACHING LABORATORY AT THE UNIVERSITY
OF WASHINGTON (REF-HUEHM71).=
ACM, SIGCSE BULL. 2, 1970, 30-33

HUEHM71 HUNT,E. KILDALL,G.
A HEATHKIT METHOD FOR BUILDING DATA MANAGEMENT PROGRAMS.=
ACM SYMP. ON ISAR, APRIL 1971

IBMDM70 IBM SYSTEMS REFERENCE LIBRARY
DATA MANAGEMENT FACILITIES OF TSS/360 (REF-KAAEU71).=
IBM REF. GC-28-2056-0, 1970, PP.22-25, 75-76

IBMPR67 IBM
PROTO-RELADES:A RESTRICTIVE NATURAL LANGUAGE SYSTEM
(REF-LODCC71).=

IBM BOSTON PROGRAMMING CENTER, TR BPC3, OCT. 3, 1967

IBMTA68 IBM
THE TIME AUTOMATED GRID SYSTEM (TAG): SALES AND SYSTEMS GUIDE (REF-TEDAR71).=
PUBLICATION NO. Y20-0358-D, 1968

INTRX70 INTREX
PROJECT INTREX SEMIANNUAL ACTIVITY REPORT (REF-KAAEU71) .=
PR-9, MARCH 1970, MIT , P.28

IVKAU68 IVERSON,K. FALKOFF,A.
APL/360: USERS MANUAL (REF-HUEHM71).=
IBM CORP., 1968

JADBI69 JACKSON,D.
BASIS FOR AN IMPROVABILITY MEASURE IN INFORMATION RETRIEVAL (REF-JADOC71).=
PROCEEDINGS OF ASIS, 6, 1969, PP. 487-495

JADCR JACKSON,D.
CLASSIFICATION, RELEVANCE AND INFORMATION RETRIEVAL (REF-JADOC71).=
IN: ADVANCES IN COMPUTERS,11, ACADEMIC PRESS. (FORTHCOMING

JADNS69 JACKSON,D.
A NOTE ON A SET OF FUNCTIONS FOR INFORMATION RETRIEVAL (REF-JADOC71).=
INFORMATION STORAGE AND RETRIEVAL, 5, 1969, PP. 27-41

JADOC71 JACKSON,D.
OPTIMAL CLASSIFICATION AND ITS CONSEQUENCES.=
ACM SYMP. ON IS&R, APRIL 1971

JEOPG65 JESPERSEN,O.
THE PHILOSOPHY OF GRAMMAR (REF-KUJQQ71).=
W.W. NORTON AND CO., INC., NEW YORK, 1965

KAADR67 KASHER,A.
DATA-RETRIEVAL BY COMPUTER: A CRITICAL SURVEY (REF-SUSMS71) (REF-KUJQQ71).=
THE GROWTH OF KNOWLEDGE, 1967, PP.292-324

KAAEU71 KATCHER,A.
EFFICIENT UTILIZATION OF LIMITED ACCESS ARCHIVAL STORAGE IN A TIME SHARED ENVIRONMENT.=
ACM SYMP. ON IS&R, APRIL 1971

KADIC67 KAHN,D.
THE CODEBREAKERS (REF-MUJDC71).=
MACMILLAN COMPANY, NEW YORK, 1967

KAHIC69 KARLGREN,H.
(CIRCULATED LETTER) (REF-DOBHF71).=
1969 INT. CONF. ON COMP. LING.

KAJL668 KALLAB,J.
A LINEAR GEOGRAPHICAL CODE FOR MANAGEMENT INFORMATION SYSTEMS (REF-MUJDC71).=
COMPUTERS AND AUTOMATION, 17, 4 (APRIL 68), 24-30

KAJPL66 KATZ,J.
THE PHILOSOPHY OF LANGUAGE (REF-KECCN71).=
NEW YORK, HARPER AND ROW, 1966

KAJRI67 KATZ,J.
RECENT ISSUES IN SEMANTIC THEORY (REF-PAMFS71).=
FOUNDATIONS OF LANGUAGE, 1967

KAJSS65 KATZ,J. FODOR,J.
THE STRUCTURE OF A SEMANTIC THEORY (REF-PAMFS71).=
READING IN THE PHILOSOPHY OF LANG. 1965

KAMEP67 KAY,M.
EXPERIMENTS WITH A POWERFUL PARSER (REF-DOBHF71) (REF-KECCN71).=
DEUXIEME CONF. INT. SUR LE TRAITEMENT AUTO. LANG., 1967

KAMLP62 KALIKOW,M.
A LONG-RANGE PROGRAM FOR MECHANIZED LEGAL AND PATENT SEARCHING CENTERS (REF-CHYFT71).=
MODERN USES OF LOGIC IN LAW PP.78-868 JUNE, 1962

KAMMS70 KAY,M. SU,S.
THE MIND SYSTEM: THE STRUCTURE OF THE SEMANTIC FILE (REF-SUSMS71).=
RM-6265/3-PR, THE RAND CORP., 1970

KEAE165 KENT,A. TAULBEE,O. (EDS.)
ELECTRONIC INFORMATION HANDLING (REF-GURER71).=
SPARTAN, 1965

KEAIM62 KENT,A.
TEXTBOOK ON MECHANIZED INFORMATION RETRIEVAL (REF-GURER71).=
INTERSCIENCE, 1962

KECCN71 KELLOGG,C. BURGER,J. DILLER,T. FOGT,K.
THE CONVERSE NATURAL LANGUAGE DATA MANAGEMENT SYSTEM: CURRENT STATUS AND PLANS.=
ACM SYMP. ON IS&R, APRIL 1971

KECDM68 KELLOGG,C.
DATA MANAGEMENT IN ORDINARY ENGLISH: EXAMPLES (REF-KECCN71).=
COMP. STUDIES IN HUMAN. & VERBAL BEHAVIOR, MOUTON VOL 1/NR.1

KECNL68 KELLOGG,C.
A NATURAL LANGUAGE COMPILER FOR ONLINE DATA MANAGEMENT (REF-KECCN71).=
PROC. 1968 AFIPS FJCC ,VOL.33, PART 1 PP.473-492

KECNL68 KELLOGG,C.

A NATURAL LANGUAGE COMPILER FOR ON-LINE DATA MANAGEMENT
(REF-PAMFS71).=
PROC. AFIPS 1968 CONF., VOL. 33, PP.473-493

KECPA70 KERPELMAN,C. (ED.)
PROPOSED AMERICAN NATIONAL STANDARD IDENTIFICATION OF STATES
.....FOR INFORMATION INTERCHANGE (REF-MUJDC71).=
COM. OF ACM, VOL. 13, # 8, 1970, PP.514-515

KEGNL68 KELLOGG,C.H.
A NATURAL LANGUAGE COMPILER FOR ON-LINE DATA MANAGEMENT
(REF-SUSMS71) .=
PROC. AFIPS 1968 FJCC, VOL. 33 , PP. 473-493

KEWET68 KENYON,W.
ERRORS IN TRANSMISSION OF COMPRESSED DATA
(REF-MUJDC71).=
PH.D. THESIS, PRINCETON UNIV. 1968

KIGAL70 KILDALL,G.
APL/B5500: THE LANGUAGE AND ITS IMPLEMENTATION
(REF-HUEHM71).=
U. W. COMP.SCI. TECH. REPORT 70-09-04, 1970

KIGEL69 KILDALL,G.
EXPERIMENTS IN LARGE SCALE COMPUTER DIRECT ACCESS STORAGE
MANIPULATION (REF-HUEHM71).=
U. W. COMP. SCI. TECH. REPORT 69-01, 1969

KIGLC63 KING,G.
AUTOMATION AND THE LIBRARY OF CONGRESS (REF-MKRS71).=
LIBRARY OF CONGRESS, DEC. 1963

KIJMA61 KISEDA,J. PETERSEN,H. SEELBACH,W. TEIG,M.
A MAGNETIC ASSOCIATIVE MEMORY (REF-GURER71).=
IBM J. RES. DEVELOP., APRIL, 1961, 106-121

KIPSA69 KING,P.
SYSTEMS ANALYSIS DOCUMENTATION: COMPUTER AIDED DATA
DIRECTORY DEFINITION (REF-TEDAR71).=
COMPUTER JOURNAL, VOL. 12, NO. 1, FEB., 1969

KNDAC68 KNUTH,D.
THE ART OF COMPUTER PROGRAMMING, VOL 1: FUNDAMENTAL
ALGORITHMS (REF-KNGBT71).=
ADDISON-WESLEY, READING, MASS., 1968

KNDOB70 KNUTH,D.
OPTIMUM BINARY SEARCH TREES (REF-KNGBT71).=
TECH REP. CS-149, STANFORD UNIV., JAN. 1970

KNGBT71 KNOTT,G.
A BALANCED TREE STORAGE AND RETRIEVAL ALGORITHM.=
ACM SYMP. ON IS&R, APRIL 1971

KNRAC69 KNUTH,R.
THE ART OF COMPUTER PROGRAMMING VOL. 1

(REF-HUEHM71).=
MENLO PK, CALIF., ADDISON-WESLEY, 1969

KOAOD66 KOHNEIM,A. WEISS,B.
AN OCCUPANCY DISCIPLINE AND APPLICATIONS,
(REF-GURER71).=
SIAM J. APPL. MATH. 14, NOV., 1966, 1266-1274

KOMAQ69 KOCHEN,M.
AUTOMATIC QUESTION-ANSWERING OF ENGLISH-LIKE QUESTIONS ABOUT SIMPLE DIAGRAMS (REF-SUSMS71).=
JOURNAL OF ACM, VOL. 16, NO. 1, JAN. 1969, PP. 26-48

KUJAQ67 KUHNS,J.
ANSWERING QUESTIONS BY COMPUTER: A LOGICAL STUDY
(REF-KUJQQ71).=
THE RAND CORPORATION, RM-5428-PR, DEC 1967

KUJAQ67 KUHNS,J.
ANSWERING QUESTIONS BY COMPUTER: A LOGICAL STUDY
(REF-DIRRD71).=
THE RAND CORPORATION, RM-5428-PR, DEC., 1967

KUJIR70 KUHNS,J.
INTERROGATING A RELATIONAL DATA FILE: REMARKS ON THE ADMISSIBILITY OF INPUT QUERIES (REF-KUJQQ71).=
THE RAND CORPORATION, R-511-PR, NOV. 1970

KUJIR70 KUHNS,J.
INTERROGATING A RELATIONAL DATA FILE: REMARKS ON THE ADMISSIBILITY OF INPUT QUERIES (REF-DIRRD71).=
THE RAND CORPORATION, R-511-PR, 1970

KUJLA69 KUHNS,J.
LOGICAL ASPECTS OF QUESTION-ANSWERING BY COMPUTER
(REF-KUJQQ71).=
THE RAND CORP., TECH. REP. P-4251, 1969

KUJQQ71 KUHNS,J.
QUANTIFICATION IN QUERY SYSTEMS.=
ACM SYMP. ON IS&R, APRIL 1971

KULCG63 KUKENHEIM,L.
CODING OF GRAMMATICAL DATA (REF-MUJDC71).=
SPRACHKUNDE UND INFOR., NO.1, 1963 PP.45-57 (GERMAN)

KUSAS67 KUNO,S.
AUTOMATIC SYNTACTIC ANALYSIS (REF-PAMFS71).=
SEMINAR ON COMPUTATION LINGUISTICS 1967

KUSMP63 KUNO,S OETTINGER,A.
MULTIPLE-PATH SYNTACTIC ANALYZER (REF-LOUCC71).=
MATH. LING. AND AUTO. TRANS., COMP. LAB., HARVARD U. 1963

LABTA66 LANGEFORS,B.
THEORETICAL ANALYSIS OF INFORMATION SYSTEMS
(REF-TEDAR71).=

2 VOL. STUDENT LITTERATUR 1966

LASA63 LANGEFORS,B.
SOME APPROACHES TO THE THEORY OF INFORMATION SYSTEMS
(REF-WACQE71).=
BIT (3), PP. 229-254, 1963

LASOS66 LAMB,S.
AN OUTLINE OF STRATIFICATIONAL GRAMMAR (REF-PAMFS71).=
GEORGETOWN UNIVERSITY PRESS, WASHINGTON, D.C., 1966

LEDFS67 LEFKOVITZ,D.
FILE STRUCTURES FOR ON-LINE SYSTEMS (REF-GURER71).=
LECTURE NITES FOR 1967 ACM SEMINAR ON FILE STRUCTURES

LEDFS69 LEFKOVITZ, D.
FILE STRUCTURES FOR ON-LINE SYSTEMS (REF-WIAFS71).=
SPARTAN BOOKS, WASHINGTON, D.C. 1969

LEDFS69 LEFKOVITZ,D.
FILE STRUCTURES FOR ON-LINE SYSTEMS (REF-HUEHM71).=
NEW YORK SPARTAN, 1969

LEDMT64 LEHMER,D.
THE MACHINE TOOLS OF COMBINATORICS (REF-GURER71).=
APPLIED COMBINATORIAL MATH., WILEY, 1964

LEMOR68 LESK,M.E.
DESIGN OF A REVISED ON-LINE INFORMATION RETRIEVAL SYSTEM
(REF-KAAEU71) .=
REP. ISR-14, INFOR. STORAGE : RETRIEVAL , CORNALL U., 1968,

LERCS67 LEVIEN,R. MARON,M.
A COMPUTER SYSTEM FOR INFERENCE EXECUTION AND DATA RETRIEVAL
(REF-DIRRD71).=
COMM. ACM, 10, 11 (NOV. 1967), 715-721

LERCS67 LEVIEN,R
A COMPUTER SYSTEM FOR INFERENCE EXECUTION AND DATA RETRIEVAL
(REF-KUJQQ71).=
COMM. ACM, V. 10, PP. 715-721, 1967

LERRD65 LEVIEN,R. MARON,M.
RELATIONAL DATA FILE: A TOOL FOR MECHANIZED INFERENCE
EXECUTION AND DATA RETRIEVAL (REF-KUJQQ71).=
THE RAND CORPORATION, RM-4793-PR, DEC. 1965

LERRD66 LEVIEN,R.
RELATIONAL DATA FILE II: IMPLEMENTATION (REF-DIRRD71).=
INFOR. RETRIE. AND A CRITICAL VIEW. 1966

LERRD69 LEVIEN,R.
RELATIONAL DATA FILE: EXPERIENCE WITH A SYSTEM FOR
PROPOSITIONAL DATA STORAGE AND INFERENCE EXEC
(REF-DIRRD71).=
THE RAND CORPORATION, RM-5947-PR, APRIL 1969

LERIM65 LEVIEN,R. MARON,M.
RELATIONAL DATA FILE: A TOOL FOR MECHANIZED INFERENCE EXECUTION AND DATA RETRIEVAL (REF-KECCN71).=
RAND CORPORATION RM-4793-PR 1965

LOCMS66 LONGYEAR,C
MEMORY STRUCTURES IN DEACON NATURAL LANGUAGE QUESTION-ANSWERING SYSTEM (REF-SUSMS71).=
P-129, GENERAL ELECTRIC CO., TEMPO, 1966

LODCC71 LOVEMAN,D.
CUE: A CUSTOMIZED SYSTEM FOR RESTRICTED, NATURAL ENGLISH.=
ACM SYMP. ON IS&R, APRIL 1971

LODIE68 LOVEMAN,D.
AN INTRODUCTION TO ELF - LIST PROCESSING IN PL/I (REF-LODCC71).=
IBM FEDERAL SYSTEMS CENTER, IM 48.68.001, JAN 22, 1968

LOEDC68 LOHSE,E. (ED.)
DATA CODE FOR CALENDAR DATE FOR MACHINE-TO-MACHINE DATA INTERCHANGE (REF-MUJDC71).=
COMM. ACM, 11, 4 (APRIL 68), 273-274

LOPRE69 LOCKEMANN,P. THOMPSON,F.
A RAPIDLY EXTENSIBLE LANGUAGE SYSTEM (REF-SUSMS71).=
PREPRINT NO. 34, INTER. CONF. ON COMPUT. LING., SWEDEN, 1969

LOPRE69 LOCKEMANN,P. THOMPSON,F.
A RAPIDLY EXTENSIBLE LANGUAGE SYSTEM: THE REL LANGUAGE PROCESSOR (REF-DOBHF71).=
1969 INT. CONF. ON COMP. LING. NO.34 ,31P.

LUHKW59 LUHN, HANS PETER.
KEY-WORD-IN-CONTEXT IDEA FOR TECHNICAL LITERATURE (KWIC INDEX). (REF-MKRS71).=
REPORT NO. RC127, IBM, YORKTOWN HEIGHTS, N. Y. 1959.

LUVMR70 LUM,V.
MULTI-ATTRIBUTE RETRIEVAL WITH COMBINED INDEXES (REF-GURER71).=
COMM. ACM 13. NOV., 1970, 660-665

MABAD67 MARRON,B. DEMAINE,P.
AUTOMATIC DATA COMPRESSION (REF-MUJDC71).=
COMM. ACM 10, (NOV 1967), 711-715

MAJDR67 MARTIN,J.
DESIGN OF REAL-TIME COMPUTER SYSTEMS (REF-TEDAR71).=
PRENTICE-HALL, ENGLEWOOD CLIFFS, N.J., 1967

MALMF69 MARTIN,D.
A MODEL FOR FILE STRUCTURE DETERMINATION FOR LARGE ON-LINE DATA FILES (REF-WIAFS71).=
SELECTED PAPERS FROM I.A.G. CONF. 1969, PP.223-245

MAMRD66 MARON,M.

RELATIONAL DATA FILE I: DESIGN PHILOSOPHY (REF-DIRRD71).=
INFOR. RETRIEVAL : A CRITICAL VIEW, 1966

MAWFC68 MAURER,W.
FILE COMPRESSION USING HUFFMAN CODING (REF-MUJDC71).=
COMPUTING METHODS IN OPTIMIZATION PROB.,VOL.2, 1969,PP.247-2

MCWIA69 MCCUSKEY,W.
TOWARD THE AUTOMATIC DESIGN OF DATA ORGANIZATION FOR LARGE
SCALE INFORMATION PROCESSING SYSTEMS (REF-WACQE71).=
PH.D. THESIS, CASE WESTERN RESERVE UNIV., JAN 1969

MECAI67 MEADOW,C.
THE ANALYSIS OF INFORMATION SYSTEMS (REF-GURER71).=
WILEY, 1967

MECMM70 MEADOW,C.
MAN-MACHINE COMMUNICATIONS (REF-GURER71).=
WILEY, 1970

MEE1M64 MENDELSON,E.
INTRODUCTION TO MATHEMATICAL LOGIC (REF-DIRRD71).=
D. VAN NOSTRAND CO., NEW YORK, 1964

MIJRD70 MINKER,J SABLE,J.D.
RELATIONAL DATA SYSTEM STUDY (REF-MKRS71) .=
RADC TR-70-180, FINAL TECH REP., REME AIR DEV. CENTER, 1970

MIMSI69 MINSKY,M. (ED.)
SEMANTIC INFORMATION PROCESSING (REF-SUSMS71).=
MIT PRESS, CAMBRIDGE, 1969

MITEP64 MITRE CORPORATION
MITRE, ENGLISH PREPROCESSOR MANUAL (REF-LOUCC71).=
THE MITRE CORP, SR-132, BEDFORD, MASS., DEC. 1964

MKRGD69 MINKER, J.
GENERALIZED DATA MANAGEMENT SYSTEMS - SOME PRESPECTIVES
(REF-MKRS71).=
U. OF MD., COMP. SCI. CENTER, TR69-101, 1969

MKRS71 MINKER,J. ROSENFELD,S.
INTRODUCTION AND PERSPECTIVES FOR THE 1971 ACM INFORMATION
STORAGE AND RETRIEVAL SYMPOSIUM .=
ACM SYMP. ON IS&R, APRIL 1971

MOCAL69 MONTGOMERY, C.
AUTOMATED LANGUAGE PROCESSING (REF-PAMFS71).=
REVIEW OF INFOR. SCI. AND TECH., VOL. 4, 1969

MOCLA69 MONTGOMERY,C.
LINGUISTICS AND AUTOMATED LANGUAGE PROCESSING
(REF-PAMFS71).=
PROC. OF INT. CONF. ON COMP. LING., NO. 41, 1969

MOJNL69 MOYNE,J.A.
INFORMATION RETRIEVAL AND NATURAL LANGUAGE

(REF-PAMFS71) .=
PROC. OF THE AMER. SOC. FOR INFOR. SCI., VOL. 6, 1969

MOJPR69 MOYNE,J.A.
A PROGRESS REPORT ON THE USE OF ENGLISH IN INFORMATION RETRIEVAL (REF-PAMFS71) .=
PROC. THE INT. CONF. ON COMP. LING., 1969

MOJTU67 MOYNE,J.
TOWARDS THE UNDERSTANDING OF NATURAL LANGUAGES BY MACHINES (REF-LODCC71).=
IBM BOSTON PROGRAMMING CENTER, TR BPC2, OCT3, 1967

MUJDC71 MULFORD,J. RIDALL,R.
DATA COMPRESSION TECHNIQUES FOR ECONOMIC PROCESSING OF LARGE COMMERCIAL FILES.=
ACM SYMP. ON IS&R, APRIL 1971

NACAD67 NATIONAL CASH REGISTER COMPANY
ACCURATELY DEFINED SYSTEMS (REF-TEDAR71).=
NATIONAL CASH REGISTER COMPANY, 1967

NAGSA68 NAGY,G.
STATE OF THE ART IN PATTERN RECOGNITION (REF-JADOC71).=
PROCEEDINGS OF IEEE, 56, 1968, PP. 836-862

NEPEE61 NEUMANN,P.
EFFICIENT ERROR-LIMITING VARIABLE-LENGTH CODES (REF-MUJDC71).=
THESIS, HARVARD UNIVERSITY, CAMBRIDGE, MASS., 1961

NIGSP69 NIBLETT,G. PRICE,N.
THE STATUS PROJECT:SEARCHING ATOMIC ENERGY LAW BY COMPUTER (REF-CHYFI71).=
REP. CLM-R101, U.K. ATOMIC ENERGY AUTH. 1969

NUJD061 NUNAMAKER,J.,JR.
ON THE DESIGN AND OPTIMIZATION OF INFORMATION PROCESSING SYSTEMS (REF-WACQE71).=
PH.D. THESIS, CASE WESTERN RESERVE UNIVERSITY, JUNE 1961

NUWAW NUGENT,W. VEGH,A.
RADC-TDR-62-13, VOL. 1 (REF-MUJDC71).=
UNKNOWN

NUWCW68 NUGENT,W.
COMPRESSION WORD CODING TECHNIQUES FOR INFORMATION RETRIEVAL (REF-MUJDC71).=
JOUR. LIBRARY AUTOMATION 1, 4 (DEC. 68), 250-260

OLYTD68 OLNEY,Y. REVERD,C. ZIFF,P.
TOWARD THE DEVELOPMENT OF COMPUTATIONAL AIDS FOR OBTAINING A FORMAL DESCRIPTION OF ENGLISH (REF-PAMFS71).=
SYSTEM DEVELOPMENT CORPORATION, SP-2766/001/00, OCT. 1968

OTMIL69 OTTEN,M. PACAK,M.

INTERMEDIATE LANGUAGES FOR AUTOMATIC LANGUAGES PROCESSING
(REF-PAMFS71).=
3RD INT. SYMP. ON COMP. AND INFOR. SCI., 1969

OVCC165 OVERHAGE,C. HARMAN,R.
PLANNING CONFERENCE ON INFORMATION TRANSFER EXPERIMENTS
(INTREX) (REF-MKRS71) .=
INTREX REP., AUG. 1965, 276P.

OVC1N70 OVERHAGE,C.
PROJECT INTREX SEMIANNUAL ACTIVITY REPORT
(REF-MUJDC71).=
PR-9, MARCH 1970, MIT, P.28

OVLLA69 OVERMYER,L.
LIBRARY AUTOMATION:A CRITICAL REVIEW (REF-SUSMS71).=
ED 034 107, DEC. 1969

PAMCM67 PACAK,M.
COMPUTATIONAL MORPHOLOGY (REF-PAMFS71).=
PAPERS IN LING. IN HONOR OF L. DOSTERT, 1967

PAMFS71 PACAK,M. PRATT,A.
THE FUNCTION OF SEMANTICS IN AUTOMATED LANGUAGE PROCESSING.=
ACM SYMP. ON IS&R, APRIL 1971

PAMHC63 PACAK,M.
HOMOGRAPHS I: CLASSIFICATION AND IDENTIFICATION
(REF-PAMFS71).=
VISTAS IN INFORMATION HANDLING, 1963

PAMHC67 PACAK,M.
HOMOGRAPHS II: CLASSIFICATION AND IDENTIFICATION
(REF-PAMFS71).=
STUDES DE LINGUISTIQUES, #5 ,1967

PEWAR57 PETERSON,W.
ADDRESSING FOR RANDOM-ACCESS STORAGE (REF-GURER71).=
IBM J. RES. DEVELOP. APRIL, 1957, 130-146

PRAAP69 PRATT,A. PACAK,M.
AUTOMATIC PROCESSING OF MEDICAL ENGLISH
(REF-PAMFS71).=
INT. CONF. ON COMP. LING., 1969

PRAIT69 PRATT,A. PACAK,M.
IDENTIFICATION AND TRANSFORMATION OF TERMINAL MORPHEMES IN
MEDICAL ENGLISH (REF-PAMFS71).=
METHOD OF INFOR. IN MEDICINE, VOL. 8, #2 , 1969

PRFSU39 PRATT,F.
SECRET AND URGENT, THE STORY OF CODES AND CIPHERS
(REF-MUJDC71) .=
BLUE RIBBON BOOKS, GARDEN CITY, NEW YORK, 1939

QURSM68 QUILLIAN,R.
SEMANTIC MEMORY (REF-KECCN71) (REF-PAMFS71) .=

SEMANTIC INFORMATION PROCESSING , MINSKY, 1968

QURTL69 QUILLIAN,R.
THE TEACHABLE LANGUAGE COMPREHENDER
(REF-PAMFS71) (REF-SUSMS71) .=
COMMUNICATIONS OF THE ACM, VOL. 12, NO. 8, AUG 1969

QUWVE60 QUINE,W.
VARIABLES EXPLAINED AWAY (REF-KECCN71).=
PROC. AMER. PHIL. SOC. VOL 1048 NO. 3, 1960 PP. 343-347

REACH64 RESNICK,A. SAVAGE,T.
THE CONSISTENCY OF HUMAN JUDGEMENTS OF RELEVANCE
(REF-JADOC71).=
AMERICAN DOCUMENTATION, 15, 1964, PP. 93-95

REHES47 REICHENBACH,H.
ELEMENTS OF SYMBOLIC LOGIC (REF-KUJQQ71).=
MACMILLAN COMPANY, NEW YORK, 1947

RONDE68 ROCHESTER,N.
THE USE OF ENGLISH FOR COMMUNICATING WITH A COMPUTER
(REF-LODCC71).=
PROC. OF THE INFORMATION SYSTEMS SYMPOSIUM, SEP. 4-6, 1968

ROPIC66 ROSENBAU,P. LUCHAK,D.
THE IBM CORE GRAMMAR OF ENGLISH (REF-LOUCC71).=
SPEC. AND UTIL. OF A TRANS. GRAM., IBM 1966

RUBIM62 RUSSELL,B.
AN INQUIRY INTO MEANING AND TRUTH (REF-KUJQQ71).=
PENGUIN BOOKS, BALTIMORE, MARYLAND, 1962

SAGAI63 SALTON,G.
AUTOMATIC INFORMATION ORGANIZATION AND RETRIEVAL
(REF-MKRS71) (REF-GURER71).=
MCGRAW-HILL, 1968

SAGAL68 SALTON,G.
AUTOMATED LANGUAGE PROCESSING (REF-PAMFS71).=
REVIEW OF INFOR. SCI. AND TECH. ,VOL. 3, 1968

SAGCE68 SALTON,G.
COMPUTER EVALUATION FOR INDEXING AND TEXT PROCESSING
(REF-JADOC71).=
J.ACM., 15, 1968, PP. 8-36

SAGDM66 SALTON,G.
DATA MANIPULATION AND PROGRAMMING PROBLEMS IN AUTOMATIC
INFORMATION RETRIEVAL (REF-GURER71).=
COMM. ACM 9. MARCH, 1966, 204-210

SAGEP70 SALTON,G.
EVALUATION PROBLEMS IN INTERACTIVE INFORMATION RETRIEVAL
(REF-SUSMS71).=
INFOR. STORAGE AND RETIEVAL, VOL. 6,1970, PP.29-44

SAGRA69 SALTON,G. LESK,M.
RELEVANCE ASSESSMENTS AND RETRIEVAL SYSTEM EVALUATION (REF-JADOC71).=
INFORMATION STORAGE AND RETRIEVAL, 4, 1969, PP. 343-359

SAGSA65 SALTON,G. LESK,M.
THE SMART AUTOMATIC DOCUMENT RETRIEVAL SYSTEM -- AN ILLUSTRATION (REF-JADOC71).=
C.ACM., 8, 1965, PP. 391-398

SCACS69 SCHREIBER,A.
COMPUTERIZED STORAGE AND RETRIEVAL OF CASE LAW WITHOUT INDEXING: THE HEBREW RESPONSA PROJECT (REF-CHYFT71).=
LAW AND COMPUTER TECHNOLOGY, PP.14-22, NOV. 1969

SCEUM63 SCHWARTZ,E.S.
A DICTIONARY FOR MINIMAL REDUNDANCY ENCODING (REF-MUJDC71).=
JACM 10, (1963) PP. 413-439

SCELE67 SCHWARTZ,E. KLEIBOEMER,A.
A LANGUAGE ELEMENT FOR COMPRESSION CODING (REF-MUJDC71).=
INFORMATION AND CONTROL, 10, 3 (MARCH 67), 315-333

SCGMK63 SCHAY,G. RAVER,N.
A METHOD FOR KEY-TO-ADDRESS TRANSFORMATION (REF-GURER71).=
IBM J. RES. DEVELOP. APRIL, 1963, 121-126

SCG67 SCHECTER, G. (ED.)
INFORMATION RETRIEVAL: A CRITICAL VIEW (REF-GURER71).=
THOMPSON, 1967

SCRST67 SCHWARCZ,R.
STEPS TOWARD A MODEL OF LINGUISTIC PERFORMANCE (REF-PAMFS71).=
MECH. TRANS. AND COMP. LING. ,NO. 10, 1967

SCRTC69 SCHARCZ,R.
TOWARDS A COMPUTATIONAL FORMALIZATION OF NATURAL LANGUAGE SEMANTICS (REF-PAMFS71).=
INT. CONF. ON COMP. LING., PREPRINT #29, 1969

SEMFD69 SENKO,M. LING,H. LUM,V. MEADOW,H. BRYMAN,M.
FILE DESIGN HANDBOOK (REF-WACQE71).=
CONTRACT REPORT AF 30602-69-C-0100, NOV 1969

SEMFO68 SENKO,M. LUM,V. OWENS,P.
A FILE ORGANIZATION EVALUATION MODEL-FOREM (REF-WACQE71).=
1968 IFIP CONGRESS PROCEEDINGS, PP. C19-23

SEMIS69 SENKO,M.
INFORMATION STORAGE AND RETRIEVAL SYSTEMS (REF-CHYFT71).=
ADVANCES IN INFORMATION SYSTEMS SCIENCE 2,PP.229-281 1969

SGHSA64 STEVENDS, M.E. GIULIANO, V.E. HEILPRIN, L.B.
STATISTICAL ASSOCIATION METHODS FOR MACHANIZED DOCUMENTATION
(REF-MKRS71) .=
NATURAL BUREAU OF STANDARDS, MISC. PUBLICATION 269, 1964

SHJML67 SHOENFIELD,J.
MATHEMATICAL LOGIC (REF-DIRR071).=
ADDICON WESLEY, READING, MASS., 1967

SHSNS69 SHAPIRO,S. WOODMANSEE,G.
A NET STRUCTURE BASED RELATIONAL QUESTION ANSWERER:
DESCRIPTION AND EXAMPLES (REF-SUSMS71).=
PROC. INT. JT. CONF. ART. INTEL., 1969, PP.325-345

SIMNL70 SIMMONS, R.F.
NATURAL LANGUAGE QUESTION-ANSWERING SYSTEMS.
(REF-SUSMS71) (REF-MKRS71) .=
CACM 13, 1, JAN. 1970, P. 15-30

SIRAE65 SIMMONS,R.
ANSWERING ENGLISH QUESTIONS BY COMPUTER: A SURVEY
(REF-MKRS71) (REF-PAMFS71) (REF-SUSMS71) .=
COMM. ACM 8, 1 (JAN8 1965) PP. 53-70

SIRCM68 SIMMONS,R. BURGER,J. SCHWARCZ,R.
A COMPUTATIONAL MODEL OF VERBAL UNDERSTANDING
(REF-SUSMS71).=
PROC. AFIPS 1968 FJCC, VOL.33, PP.441-456

SIRCM68 SIMMONS,R. BURGER,J. SCHWARCZ,R.
A COMPUTATIONAL MODEL OF VERBAL UNDERSTANDING
(REF-PAMFS71).=
PROC. 1968 AFIPS CONF., VOL. 33

SIRLA68 SIMMONS,R.
LINGUISTIC ANALYSIS OF CONSTRUCTED RESPONSES IN CAI
(REF-PAMFS71).=
UNIVERSITY OF TEXAS, COMPUTATION CENTER, TNN-86, 1968

SIRNL70 SIMMONS,R.
NATURAL LANGUAGE QUESTION ANSWERING SYSTEMS (REF-PAMFS71).=
COMMUNICATIONS OF THE ACM, VOL. 13, NO. 1, JAN. 1970

SLMMS69 SLAE,M.
MARRANOS SEARCH IN THE RIVASH RESPONSA (IN HEBREW)
(REF-CHYFI71).=
BAR-ILAN YEARBOOK 7-8, PP.397-419 1969-70

SLMOR70 SLAE,M.
THE OCCULT IN THE RIVASH RESPONSA (IN HEBREW)
(REF-CHYFI71).=
SHANA B-SHANAH 5731, PP.226-236 1970

SNMMV70 SNYDERMAN,M. HUNT,B.
THE MYRIAD VIRTUES OF TEXT COMPACTION (REF-MUJUC71) .=
DATAMATION, 16, 16 (DEC. 1970) P.36

SPJIS65 SPIEGEL,J. WALKER,D. (EDS)
INFORMATION SYSTEMS SCIENCES: PROCEEDING OF THE SECOND CONGRESS (REF-GURER71).=
SPARTAN, 1965

SPKCA67 SPARCK JONES,K. JACKSON,D.
CURRENT APPROACHES TO CLASSIFICATION AND CLUMP-FINDING AT THE CLRU (REF-JADOC71).=
COMPUTER JOURNAL, 10, 1967, PP. 29-37

STISK63 SUTHERLAND,I.
SKETCHPAD, A MAN-MACHINE GRAPHICAL COMMUNICATION SYSTEM (REF-HUEHM71) .=
PROC. 1963 AFIPS SJCC, PP.327-346

STMAI70 STEVENS,M.E.
AUTOMATIC INDEXING: A STATE OF THE ART REPORT (REF-MKRS71).=
NAT. BUR. OF STANDARDS, NBS MONO. 91, 1970, 290P.

STRIT68 STOCKWELL,R. SCHACHTER,P. PARTEE,B.
INTEGRATION OF TRANSFORMATIONAL THEORIES OF SYNTAX (REF-KECCN71).=
USAF ELEC. SYS. DIV., OCT. 1968 VOL. I, II.

STRIT68 STOCKWELL,R.P. SCHACHTER,P. PARTEE,B.H.
INTEGRATION OF TRANSFORMATIONAL THEORIES ON ENGLISH SYNTAX (REF-PAMES71) .=
VOL. I, II, ESD-TR-68-419, UCLA , OCT. 1968

SUSUR69 SU,S. HARPER,K.
A DIRECTED RANDOM PARAGRAPH GENERATOR (REF-SUSMS71).=
PREPRINT #13, INT. CONF. ON AOMP. LING., 1969

SUSMS71 SU,S.
MANAGING SEMANTIC DATA IN AN ASSOCIATIVE NET.=
ACM SYMP. ON IS&R, APRIL 1971

SUSST69 SU,S.
A SEMANTIC THEORY BASED UPON INTERACTIVE MEANING (REF-SUSMS71).=
TECH REP. NO.68, UNIV. OF WISCONSIN, 1969

SWJEI67 SWETS,J.
EFFECTIVENESS OF INFORMATION RETRIEVAL METHODS (REF-JADOC71).=
REP. #1499, BOLT, BERANEK & NEWMAN 1967

TAACR41 TARSKI,A.
ON THE CALCULUS OF RELATIONS (REF-KUJQQ71).=
J. SYMBOLIC LOGIC, V.6, PP. 73-89, 1941

TAMNU66 TARTER,M. KRONMAL,R.
NON-UNIFORM KEY DISTRIBUTION AND ADDRESS CALCULATION SORTING (REF-GURER71).=
PROC. OF 21ST ACM NAT. CONF., 1966

TANP65 TAUBE,M.
A NOTE ON THE PSEUDO-MATHEMATICS OF RELEVANCE
(REF-JADOC71).=
AMERICAN DOCUMENTATION, 16, 1965, PP. 69-72

TEDAR71 TEICHROEW,D.
AN APPROACH TO RESEARCH IN FILE ORGANIZATION.=
ACM SYMP. ON IS&R, APRIL 1971

TEDMD69 TEICHROEW,D.
METHODOLOGY FOR THE DESIGN OF INFORMATION PROCESSING SYSTEMS
(REF-TEDAR71).=
PROCEEDINGS OF FOURTH AUSTRALIAN COMP. CONG., AUG., 1969

TEDRS69 TEICHROEW,D.
A REVIEW OF SYSTEM DESIGN TECHNIQUES (REF-WACQE71).=
ISDOS WORKING PAPER 17, UNIVERSITY OF MICHIGAN, JAN 1969

THFRE69 THOMPSON,F. LOCKEMANN,P. DOSTERT,G. DEFERILL,R.
REL: A RAPIDLY EXTENSIBLE LANGUAGE SYSTEM
(REF-DOBHF71).=
PROC. 24TH NATIONAL ACM CONFERENCE, AUG. 1969, PP.399-419

THJPS69 THORNE,J.
A PROGRAM FOR THE SYNTACTIC ANALYSIS OF ENGLISH SENTENCES
(REF-PAMFS71).=
COMMUNICATIONS OF THE ACM, VOL. 12, NO. 8, AUGUST 1969

TOJAA66 TOYODA,J. TEZUKA,Y. KASHARA,Y.
ANALYSIS OF THE ADDRESS ASSIGNMENT PROBLEM FOR CLUSTERED
KEYS (REF-GURER71).=
J. ACM 13, OCT., 1966, 526-532

TUSD63 TURNBURKE,V.P.,JR.
SEQUENTIAL DATA PROCESSING DESIGN (REF-TEDAR71).=
IBM SYSTEMS JOURNAL, MARCH 1963

VAPCC70 VAUGHN,P.
CAN COBOL COPE (REF-TEDAR71).=
DATAMATION, SEPT 1, 1970, PP. 42-46

VEJED VERHOEFF,J.
ERROR DETECTING AND CORRECTING CODES FOR THE DECIMAL NUMBER
SYSTEM (REF-MUJDC71).=
PROC. INT. SYMP. ON AUTO. OF POPULATION REG. SYS., VOL.1,

VIBOR65 VICKERY,B.
ON RETRIEVAL SYSTEM THEORY (REF-GURER71).=
BUTTERWORTH, 1965

WAC1R65 WALSTON,C.
INFORMATION RETRIEVAL (REF-GURER71).=
ADVANCES IN COMPUTERS, ACSEMIC PRESS, 1965

WACQE71 WANG,C. LUM,V.
QUANTITATIVE EVALUATION OF DESIGN TRADEOFFS IN FILE SYSTEMS
.=

WADCL69 WALKER,D.
COMPUTATIONAL LINGUISTIC TECHNIQUES IN AN ON-LINE SYSTEM FOR TEXTUAL ANALYSIS (REF-SUSMS71).=
PREPRINT #63, INT. CONF. ON COMP. LING., 1969

WEJEC66 WEIZENBAUM,J.
ELIZA:A COMPUTER PROGRAM FOR THE STUDY OF NATURAL LANGUAGE COMMUNICATION BETWEEN MAN AND MACHINE (REF-LODCC71).=
COMM ACM 9, 1 (JAN., 1966) P. 36-45

WEPCR71 WESTON,P. TAYLOR,S.
CYLINDERS: A RELATIONAL DATA STRUCTURE (REF-SUSMS71).=
ACM 1971 SYMP.

WEUES66 WEINREICH,U.
EXPLORATIONS IN SEMANTIC THEORY (REF-PAMFS71).=
CURRENT TRENDS IN LANGUISTICS, 1966

WHAPM50 WHITEHEAD,A. RUSSELL,B.
PRINCIPIA MATHEMATICA (SECOND EDITION) (REF-KUJQQ71).=
CAMBRIDGE UNIV. PRESS, 1950

WIAFS71 WINKLER,A. DALE,A.
FILE STRUCTURE DETERMINATION.=
ACM SYMP. ON IS&R, APRIL 1971

WIAMC70 WINKLER,A.
A METHODOLOGY FOR COMPARISON OF FILE ORGANIZATION AND PROCESSING PROCEDURES FOR HIERARCHICAL STORAGE STRUCTURES (REF-WIAFS71).=
PH.D. DISSERTATION, UNIV. OF TEXAS, 1970

WILPI53 WITTGENSTEIN,L.
PHILOSOPHICAL INVESTIGATIONS (REF-PAMFS71).=
THE MACMILLAN CO., 1953

WIPIF60 WINDLEY,P.
TREES, FORESTS, AND REARRANGING (REF-KNGBT71).=
C. J., VOL. 3, NO. 2, PP 84-88; JULY 1960

WIRCR62 WILSON,R.
COMPUTER RETRIEVAL OF CASE LAW (REF-CHYFT71).=
SOUTHWESTERN LEGAL J. 16, PP. 409-438, 1962

WOWPS68 WOODS,W.
PROCEDURAL SEMANTICS FOR A QUESTION-ANSWERING MACHINE (REF-PAMFS71) (REF-SUSMS71) .=
PROC. AFIPS 1968 FJCC, VOL. 33, PP.457-471

WOWSI67 WOODS,W.
SEMANTIC INTERPRETATION OF ENGLISH QUESTIONS ON A STRUCTURED DATA BASE (REF-PAMFS71).=
MATH. LING. AND AUTO. TRANS.,REP. NSF-19, 1967

WOWIN70 WOODS,W.

TRANSITION NETWORK GRAMMARS FOR NATURAL LANGUAGE ANALYSIS
(REF-KECCN71) (REF-DORHF71) .=
COMM. OF THE ACM, VOL. 13, PP 591-606, 1970

WRMML60 WRIGHT,M.
MECHANIZING A LARGE INDEX; APPENDIX: THE SOUNDEX CODE
(REF-MUJDC71).=
THE COMPUTER JOURNAL, 3 (JULY 1960), P. 83

YNCMI62 YNGVE, V. H.
COMIT AS AN IR LANGUAGE (REF-MKRS71).=
COMM. ACM 5:1 (JAN. 1962) P. 19-28.

YOJAF59 YOUNG,J. KENT,H.
ABSTRACT FORMULATION OF DATA PROCESSING PROBLEMS
(REF-IEDAR71).=
J. OF IND. ENGR., NOV.-DEC. 1958, PP.471-479

YOM1S69 YOVITS,M.
INFORMATION SCIENCE: TOWARD THE DEVELOPMENT OF A TRUE
SCIENTIFIC DISCIPLINE (REF-SUSMS71).=
AMERCIAN DOCUMENTION, OCT.1969, PP. 369-376

@BRKPT PRINT$